

# Sesión 01

## Introducción a Microservicios y Net

Instructor:

**ERICK ARÓSTEGUI**

[earostegui@galaxy.edu.pe](mailto:earostegui@galaxy.edu.pe)



# 6

## NET

### MICROSERVICES ARCHITECTURE



# ÍNDICE

01

Introducción a Microservicios (historia, evolución, ventajas, desventajas y tendencias)

---

02

Principales patrones de microservicios y su desarrollo en el curso

---

03

Desarrollo de la arquitectura base de los microservicios, contenerización (Docker)

---

04

Mi primer microservicio en NET 6.

---

01

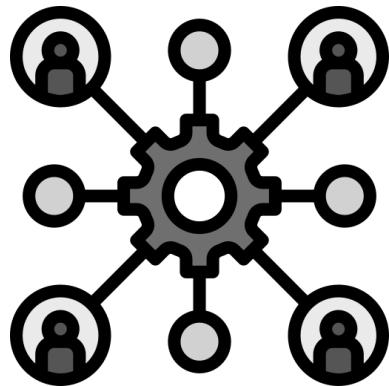


# Introducción a Microservicios (historia, evolución, ventajas, desventajas y tendencias)

# ¿Qué es arquitectura de aplicación?

# → Introducción a Microservicios

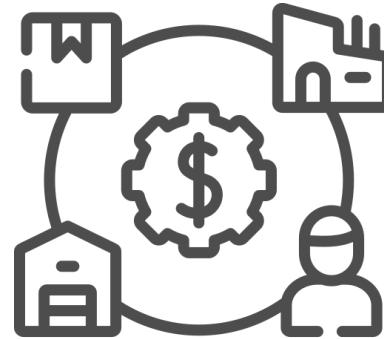
## ¿Qué es una empresa?



Una sola organización



Partes de una gran organización (como una unidad de negocios)



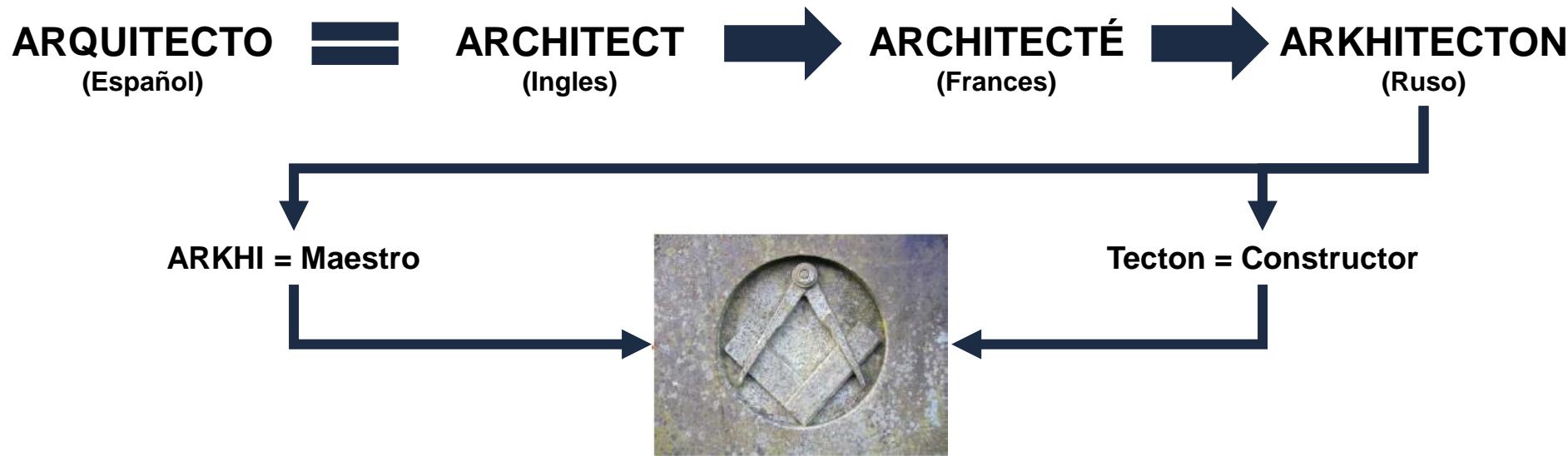
Una colección de organizaciones que colaboran en una cadena de valor.



La palabra "Empresa" cubre un amplio espectro de entidades organizacionales

# → Introducción a Microservicios

## Análisis etimológico de la arquitectura



**La arquitectura es el oficio de los maestros constructores**

# → Introducción a Microservicios

## Análisis etimológico de la arquitectura



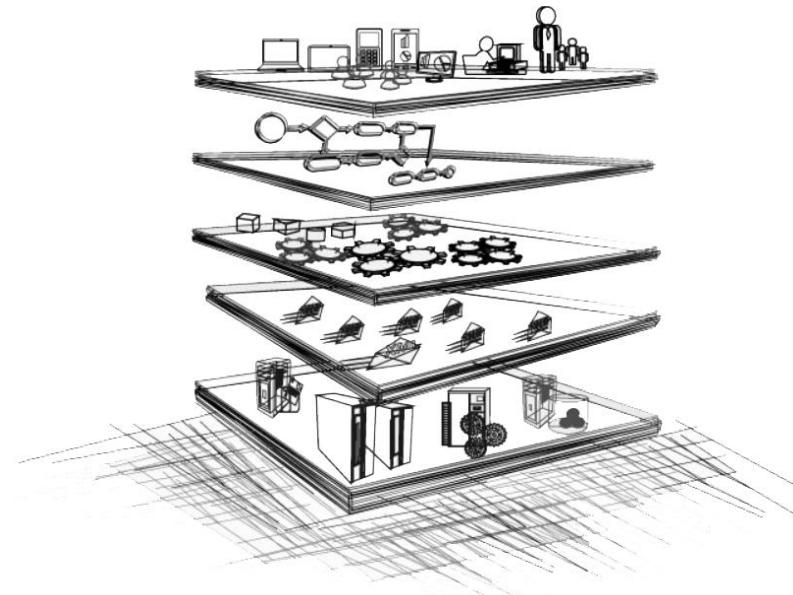
La arquitectura empresarial se puede interpretar como “**el arte de crear un plan de ejecución para la empresa**”

# → Introducción a Microservicios

## ¿Por qué necesitamos arquitectura empresarial?

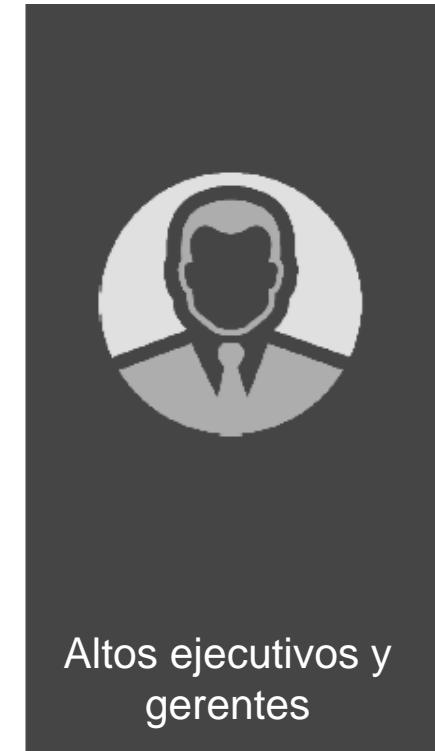
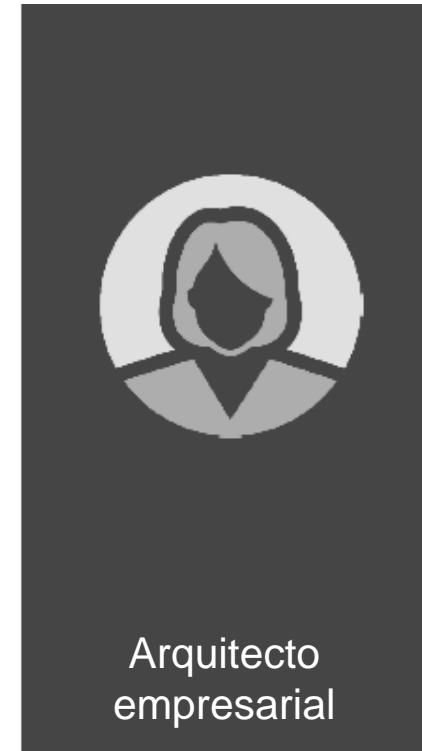
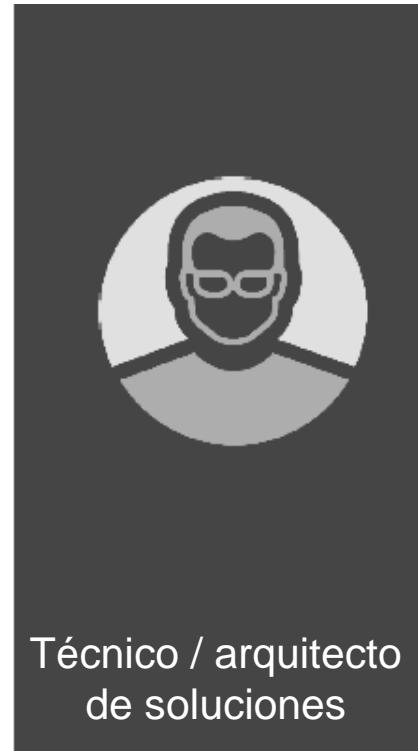
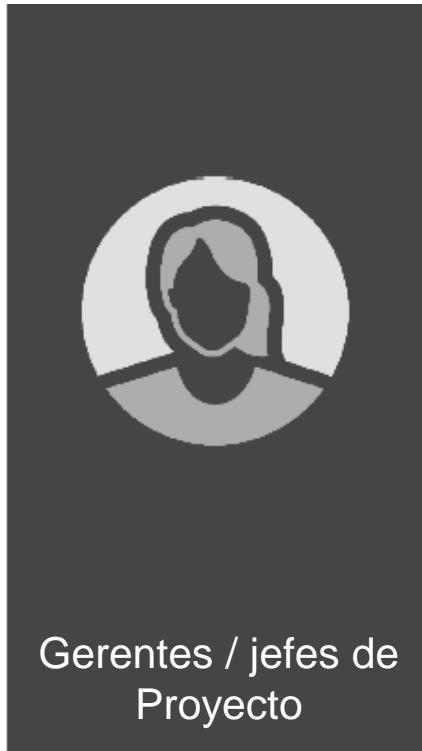


Históricamente, surgió como un mecanismo para manejar la complejidad de la implementación de sistemas de TI



En el camino, el papel de EA se transformó para abordar la arquitectura de toda la empresa en lugar de solo los componentes de TI

## Interesados



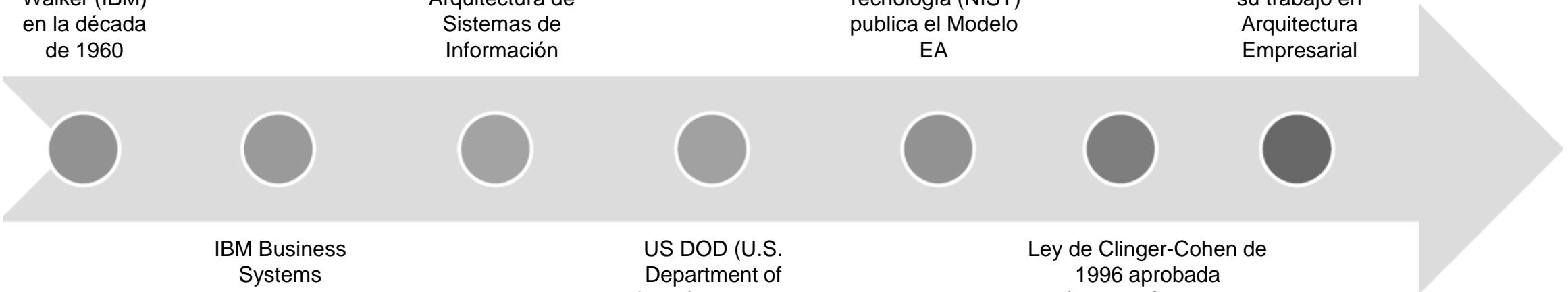
## Arquitectura Empresarial línea de tiempo

Primeras obras de Duane P Walker (IBM) en la década de 1960

1987 John Zachman publica : Framework para la Arquitectura de Sistemas de Información

1989 El Instituto Nacional de Estándares y Tecnología (NIST) publica el Modelo EA

2006 Jean Ross (MIT Sloan School) publica su trabajo en Arquitectura Empresarial



IBM Business Systems Planning (BSP) de 1970 y principios de 1980

US DOD (U.S. Department of Defense) trabaja en el Marco de Arquitectura Técnica para la Gestión de la Información (TAFIM)

Ley de Clinger-Cohen de 1996 aprobada (legislación sobre adquisición y gestión de la tecnología de la información por parte de la Administración federal)

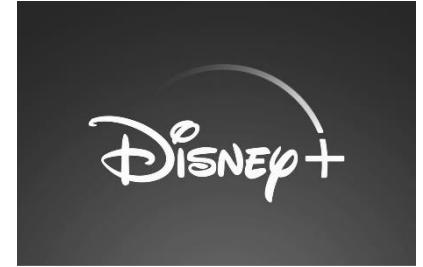
# → Introducción a Microservicios

## Realidad actual



La realidad empresarial actual es significativamente diferente de la de los años sesenta y setenta  
La naturaleza de los problemas, las oportunidades y el entorno empresarial que enfrentan las empresas hoy en día son significativamente diferentes a las décadas anteriores.

## Realidad actual



**Las empresas se han vuelto cada vez más dinámicas y complejas.**

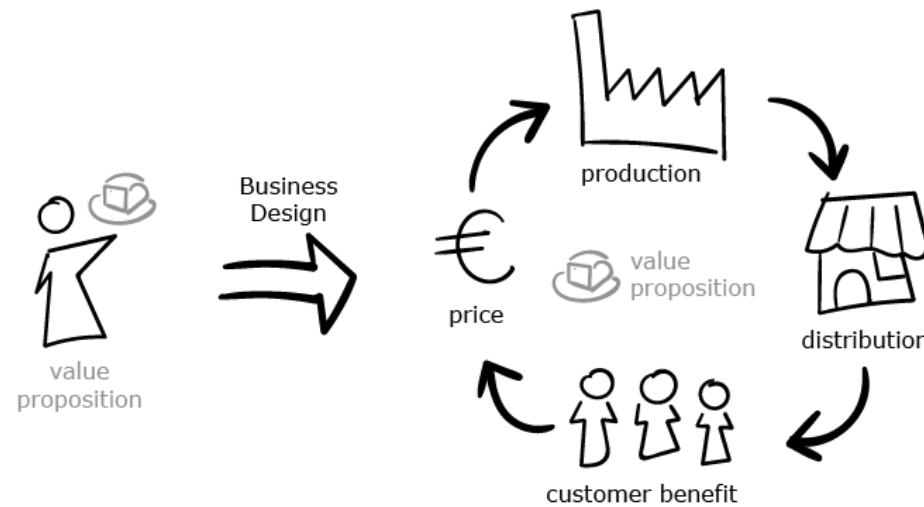
El papel de la TI dentro de las empresas tomó muchos giros inesperados en el camino.

La mayoría de las empresas ven la TI como una competencia básica fundamental

El ritmo del cambio solo se está acelerando

# → Introducción a Microservicios

## Arquitectura empresarial



“La Arquitectura empresarial es una disciplina que permite **diseñar la empresa de manera consciente y deliberada**, en lugar de dejar que ocurra al azar.”



**“El diseño se basa en la visión empresarial**, la intención estratégica y los conocimientos sobre el funcionamiento de la empresa.”

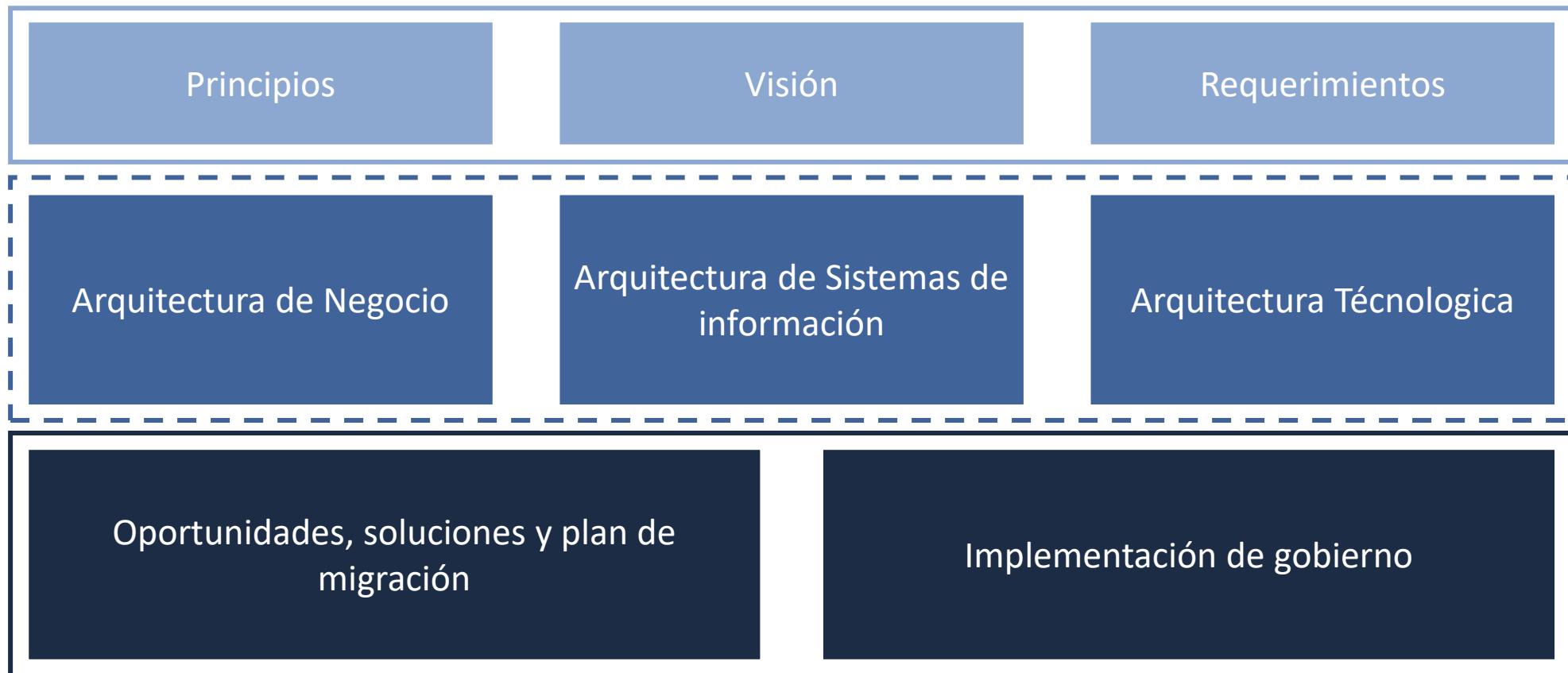
# → Introducción a Microservicios

**La arquitectura empresarial adopta vistas tanto atómicas como holísticas**



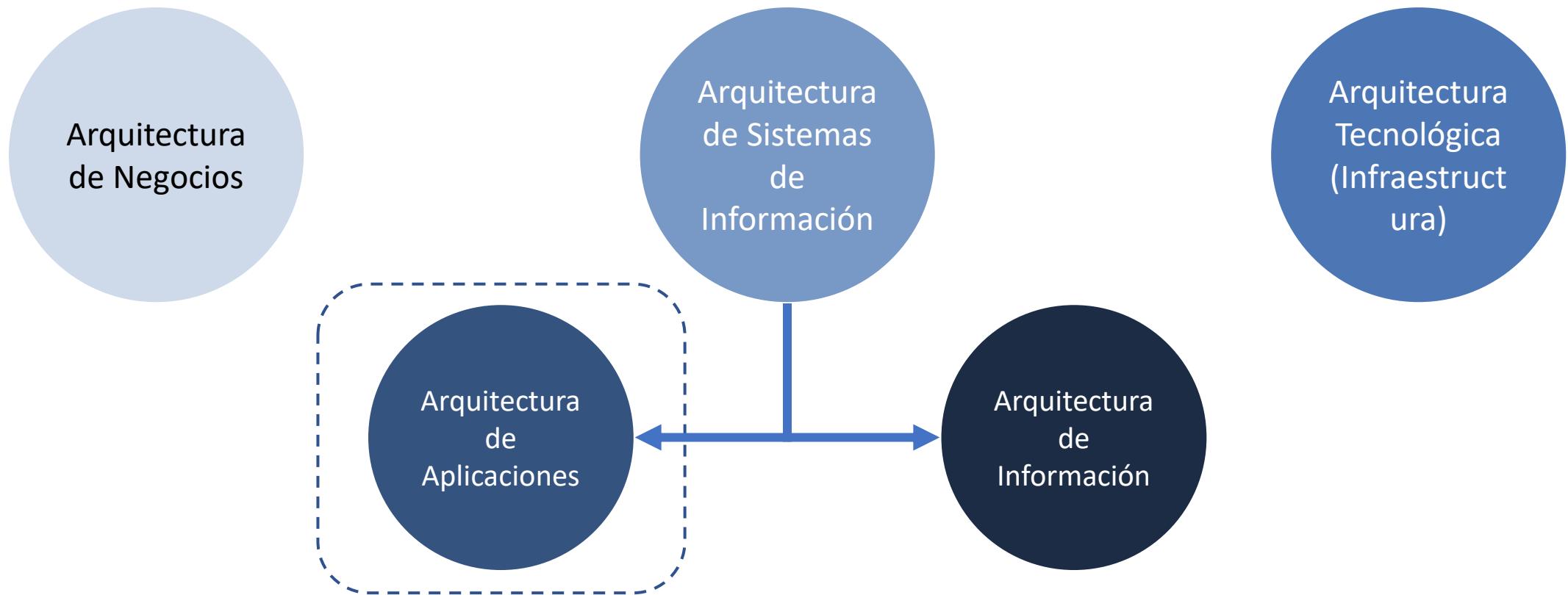
# → Introducción a Microservicios

## Contenido de la arquitectura empresarial



# → Introducción a Microservicios

## Dominios de arquitectura empresarial



# → Introducción a Microservicios

**En las últimas décadas, las empresas han invertido mucho en aplicaciones comerciales en todos los sectores industriales.**



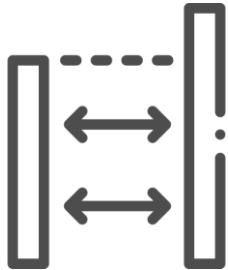
# → Introducción a Microservicios



El arquitecto de aplicaciones empresariales crea una hoja de ruta de cartera de aplicaciones de estado objetivo, teniendo en cuenta:

**Costo total de cambio  
Rendimiento de las inversiones  
Riesgos y camino de menor resistencia.**

## El estado objetivo podría incluir



Brechas identificadas en las capacidades de la aplicación



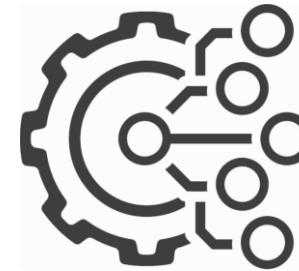
Decisión de retirar el envejecimiento y las aplicaciones de bajo valor



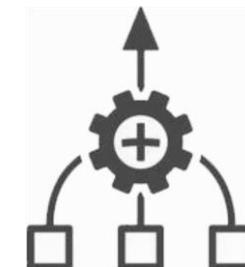
Modernización de aplicaciones heredadas pero de alto valor



Eliminar la redundancia



Estandarización en plataforma tecnológica común



Consolidando aplicaciones

# → Introducción a Microservicios

## ¿Qué es la arquitectura de software?

Cuando las personas en la industria del software hablan de "**arquitectura**", se refieren a una noción definida de los aspectos más importantes del **diseño interno de un sistema de software**.

Una buena arquitectura es importante, de lo contrario se vuelve más lento y más caro agregar nuevas capacidades en el futuro.

<https://martinfowler.com/architecture/>



# → Introducción a Microservicios

## Arquitectura de aplicaciones

Las decisiones importantes en el desarrollo de software varían con la escala del contexto en el que estamos pensando. Una escala común es la de una aplicación, por lo tanto, "**arquitectura de aplicación**".

El primer problema con la definición de la arquitectura de aplicación es que no hay una definición clara de lo que es una aplicación. Mi opinión es que las aplicaciones son una construcción social:

Un cuerpo de código que los desarrolladores ven como una sola unidad

Un grupo de funcionalidades que los clientes empresariales ven como una sola unidad

Una iniciativa que los que tienen el dinero ven como un presupuesto único

Una definición tan suelta conduce a muchos tamaños potenciales de una aplicación, que varían de unas pocas a unos pocos cientos de personas en el equipo de desarrollo. (Te darás cuenta de que miro el tamaño como la cantidad de personas involucradas, que creo que es la forma más útil de medir esas cosas.) La diferencia clave entre esto y la arquitectura empresarial es que hay un grado significativo de propósito unificado en torno a la construcción social.

<https://martinfowler.com/architecture>



# → Introducción a Microservicios

## Considere la complejidad de :

Codificación a una interfaz

Servicios

Pruebas automatizadas

Domain Driven Design

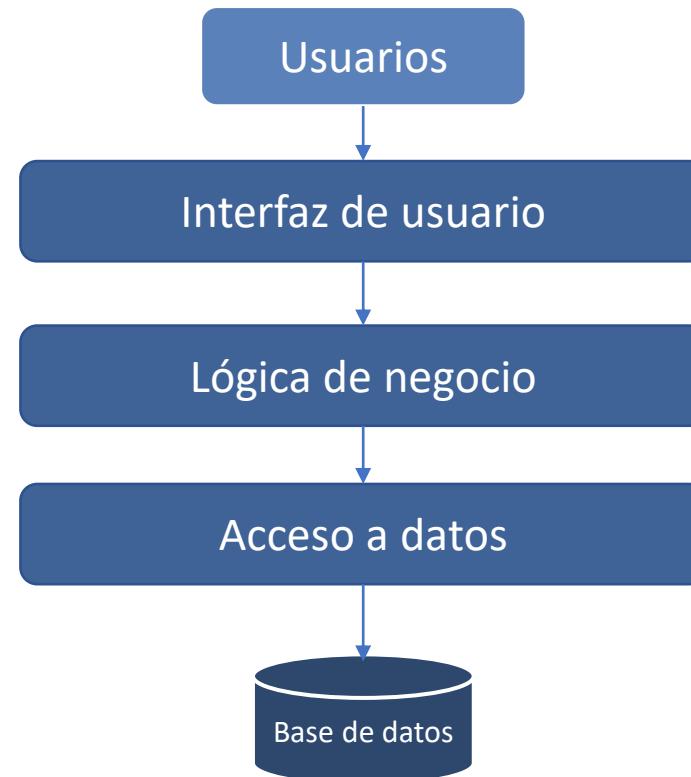
Acceso a datos

Arquitectura en capas

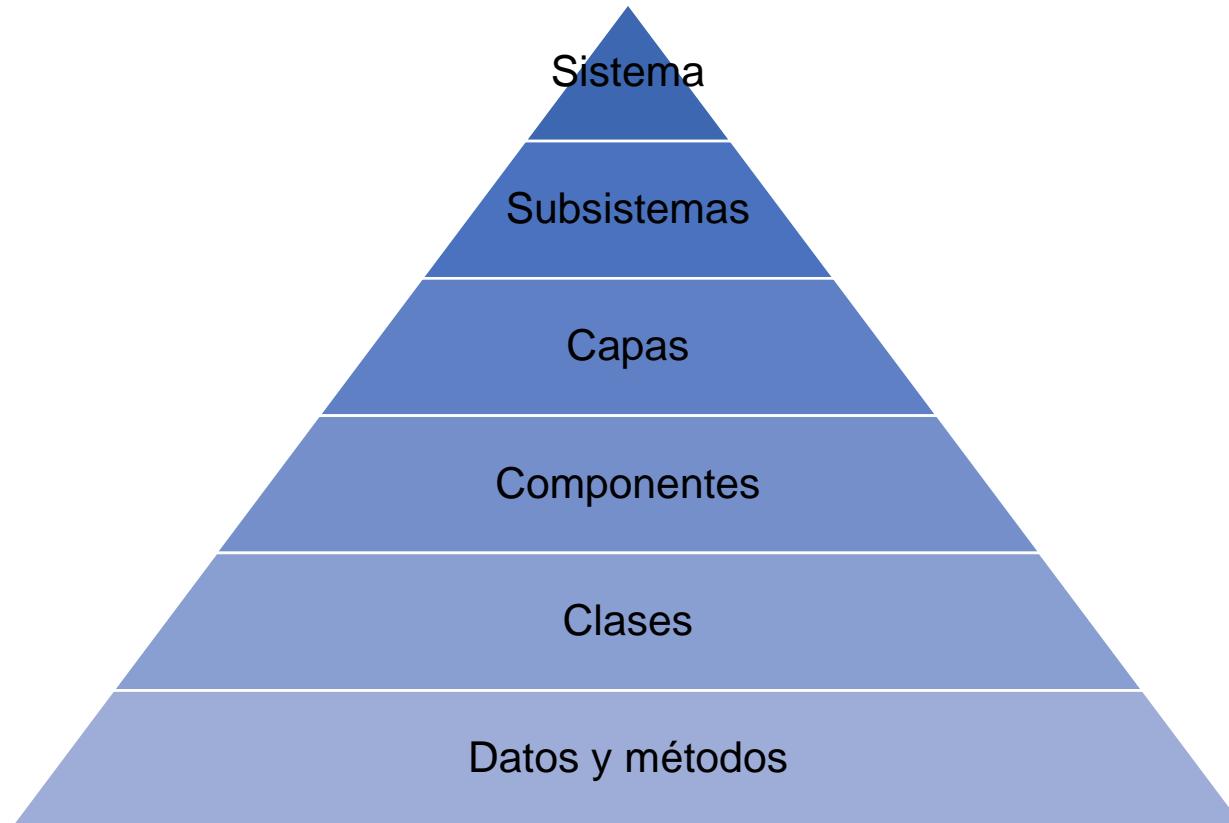
# → Introducción a Microservicios

## ¿Qué es la arquitectura de software?

Estructura a alto-nivel de capas, componentes y la relación entre estas.



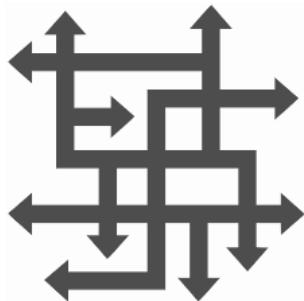
## Niveles de abstracción arquitectónica



## Arquitectura desordenada vs limpia



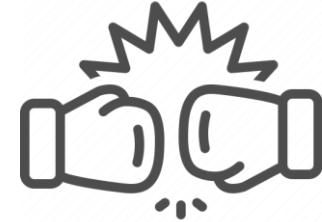
## ¿Cuándo una arquitectura es mala?



Compleja



Incoherente



Rígido



Frágil



Inestable



Insostenible

## ¿Cuándo una arquitectura es buena?



Sencilla



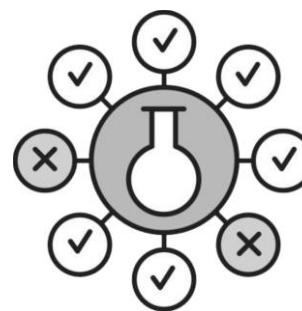
Comprendible



Flexible



Emergente



Testable



MANTENIBLE

# → Introducción a Microservicios

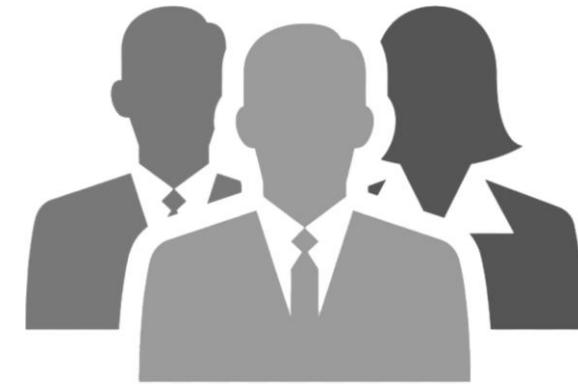
## ¿Cuál es el objetivo de una arquitectura?



Maquina



Arquitecto



Usuarios

## ¿Cuál es el objetivo de una arquitectura?



Maquina



Arquitecto

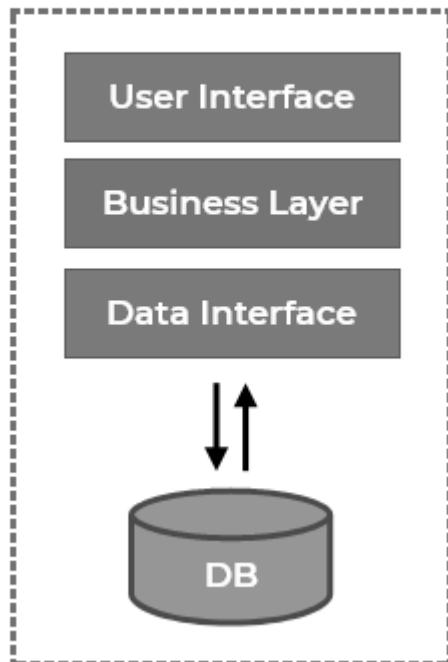


Usuarios

# Aplicaciones monolíticas

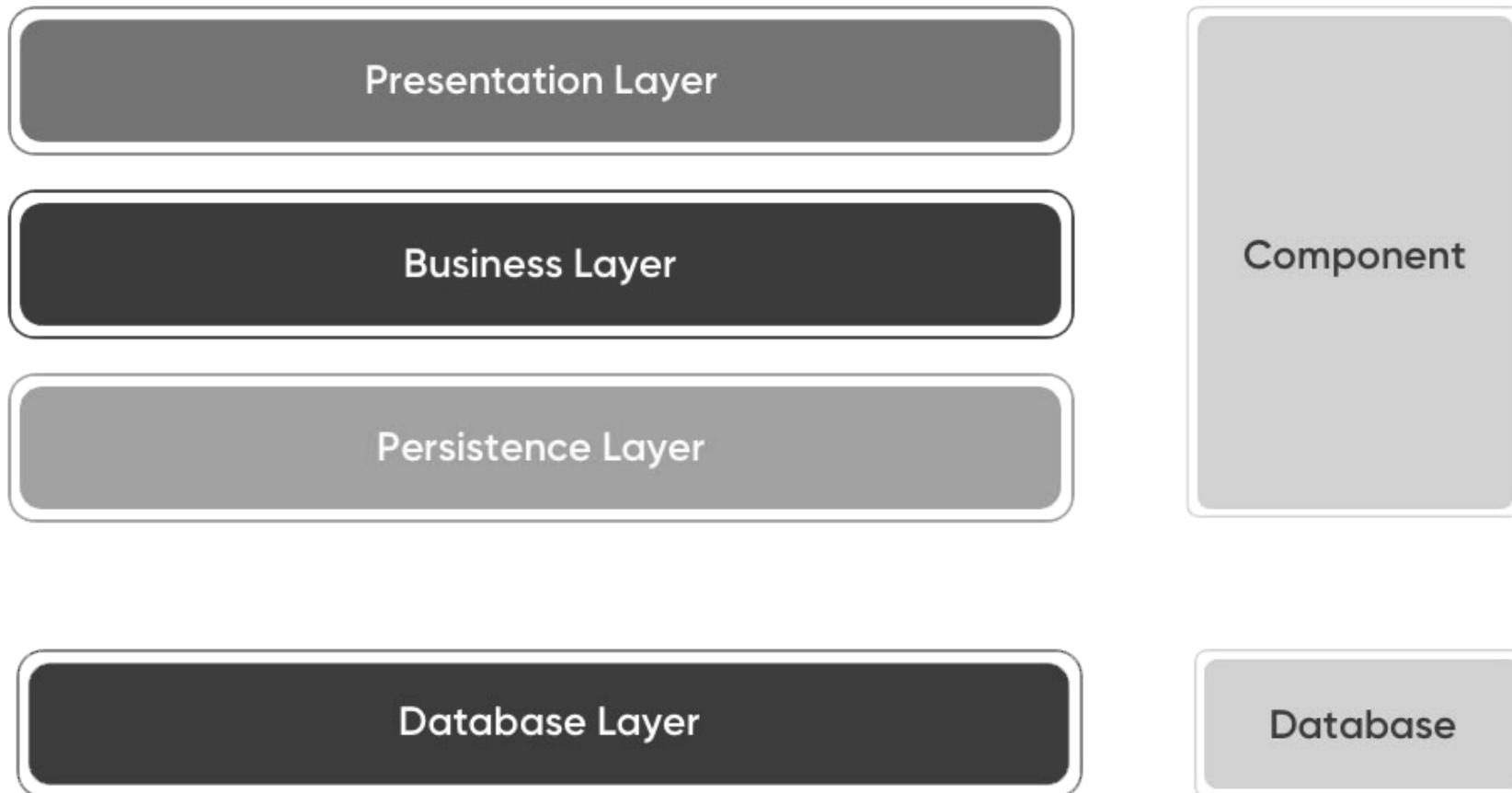
## Aplicación monolítica

Monolithic Architecture



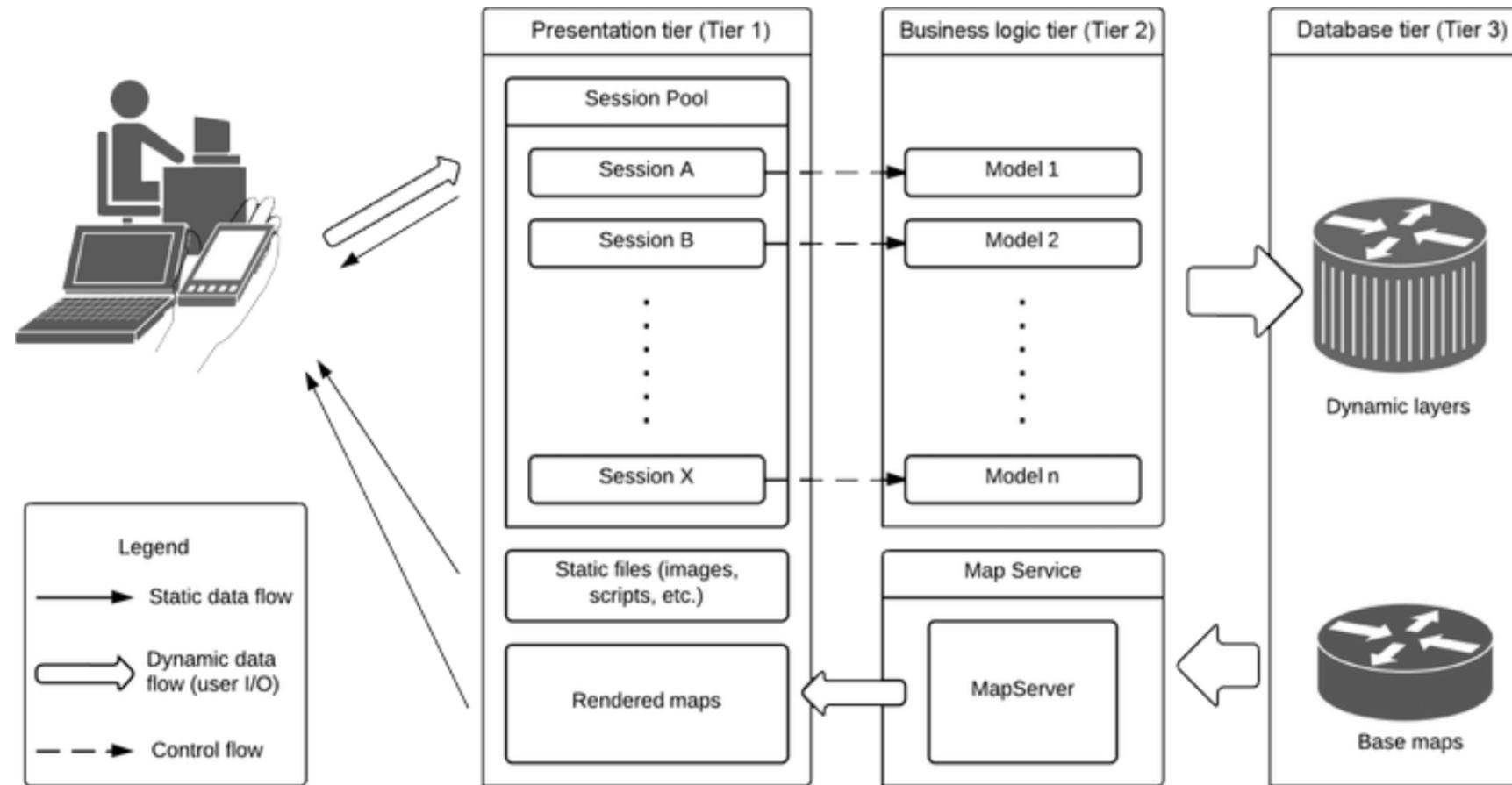
Una aplicación de software de un solo nivel en el que la interfaz de usuario y el código de acceso a datos se combinan en un solo programa desde una sola plataforma

## Acoplamiento vertical (Capas)

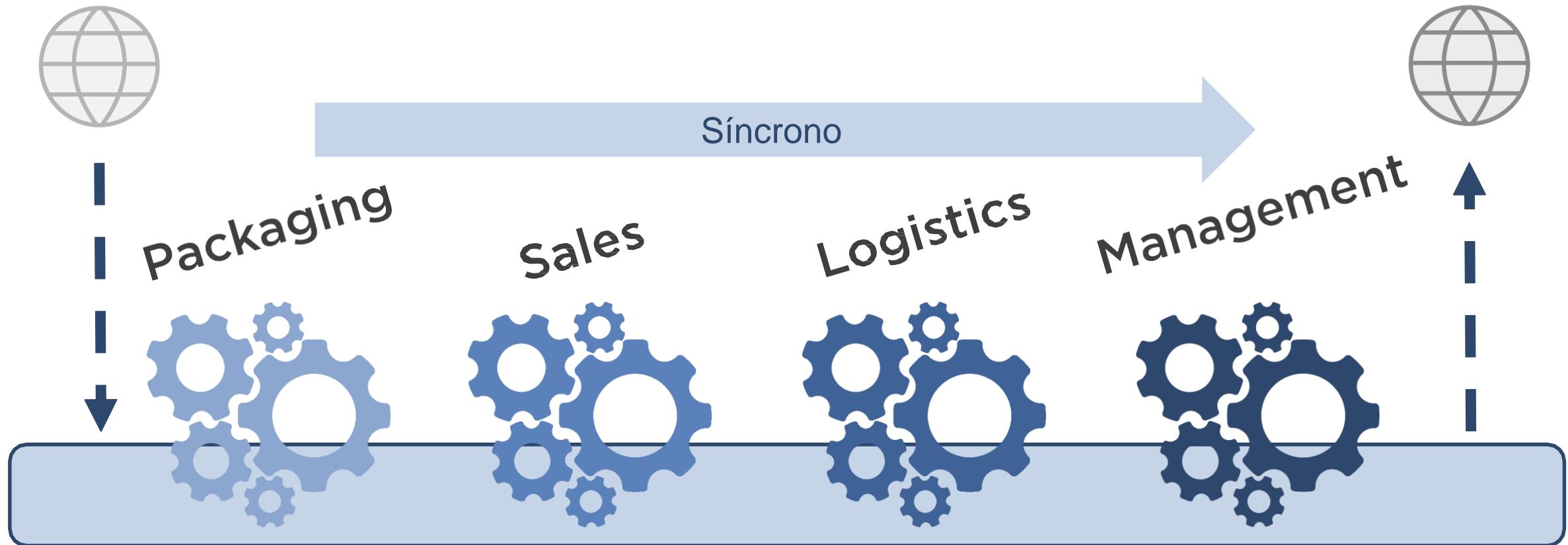


# → Introducción a Microservicios

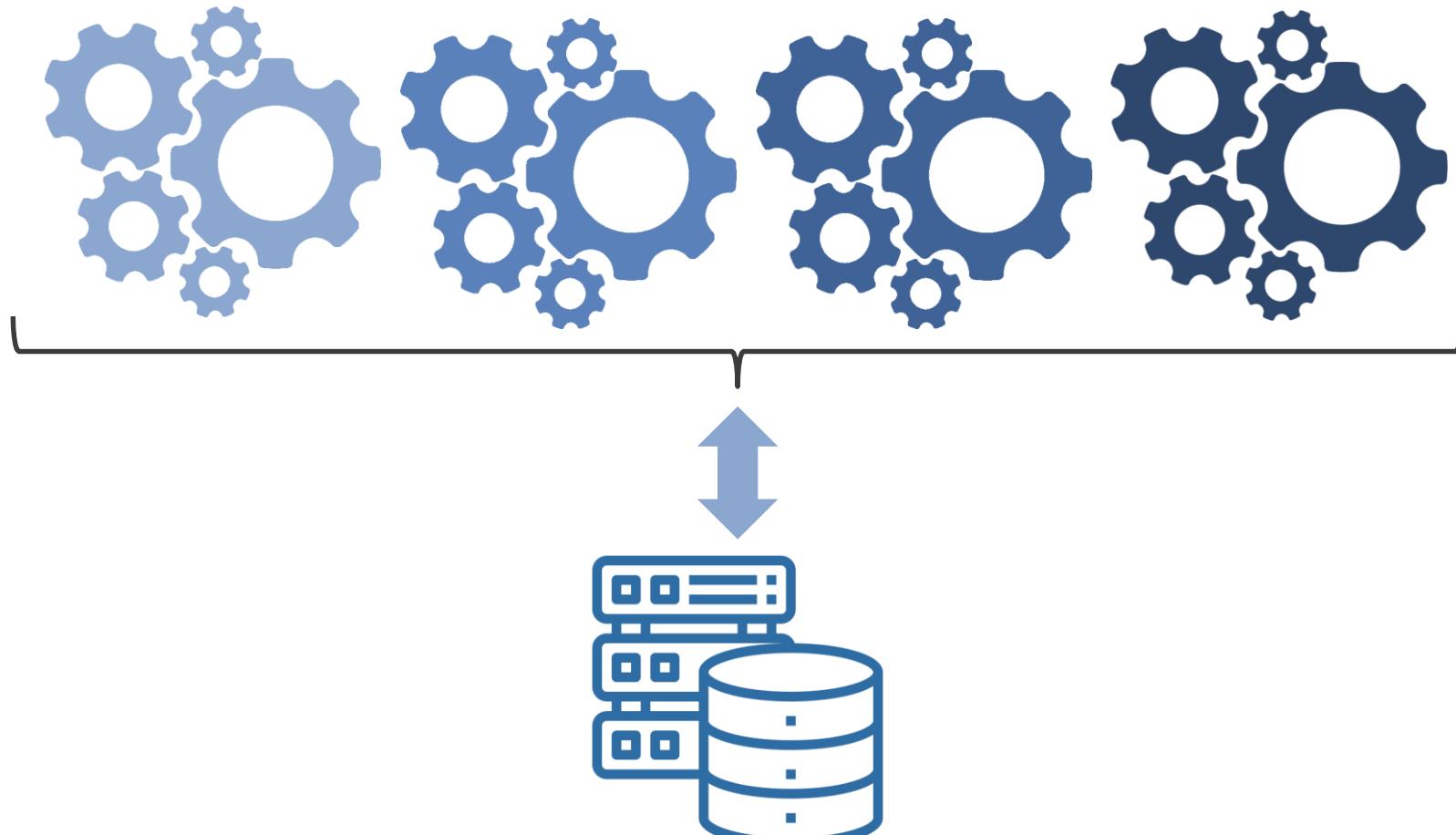
## Acoplamiento horizontal (Niveles)



## Manejo de solicitudes



## Manejo de solicitudes



## Beneficios de las aplicaciones monolíticas

Fácil de desplegar

Bien conocidas

Sin dependencias externas

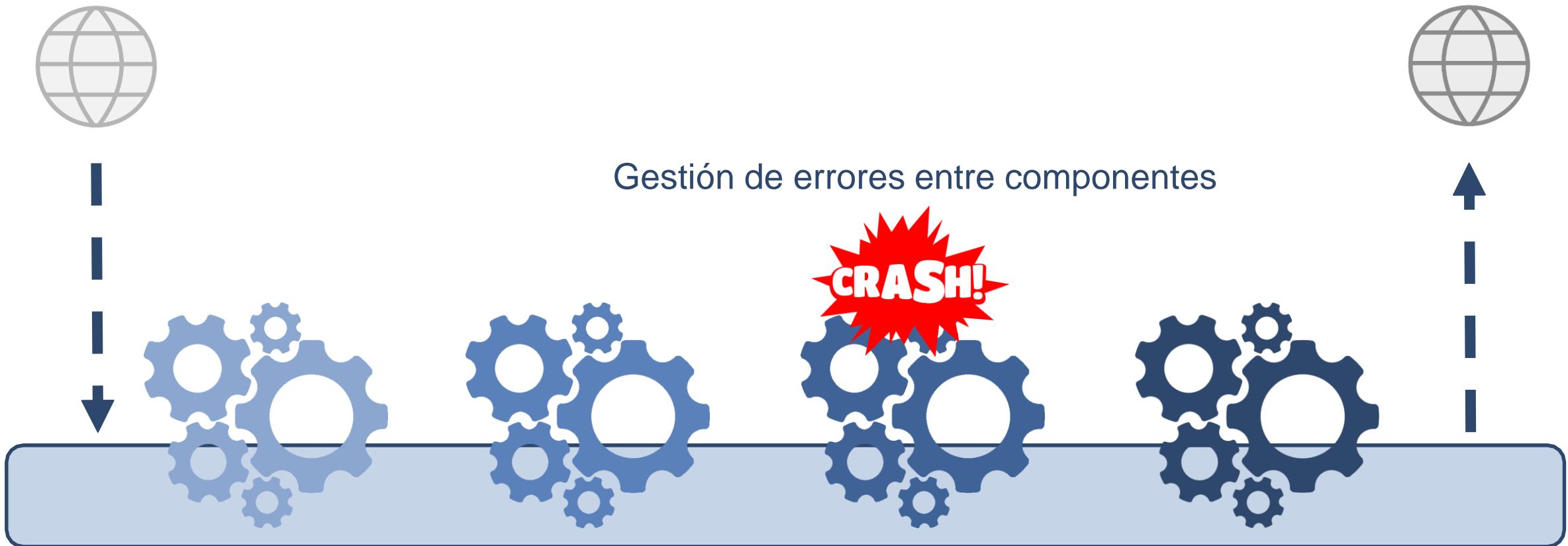
Amigable con los IDE's



## Posibles problemas



## Posibles problemas



## Posibles problemas



# → Introducción a Microservicios

## Desventajas de las aplicaciones monolíticas

Complejo y difícil de mantener

Tiende a complicarse con el tiempo

Despliegue trabajoso (Coordinación)

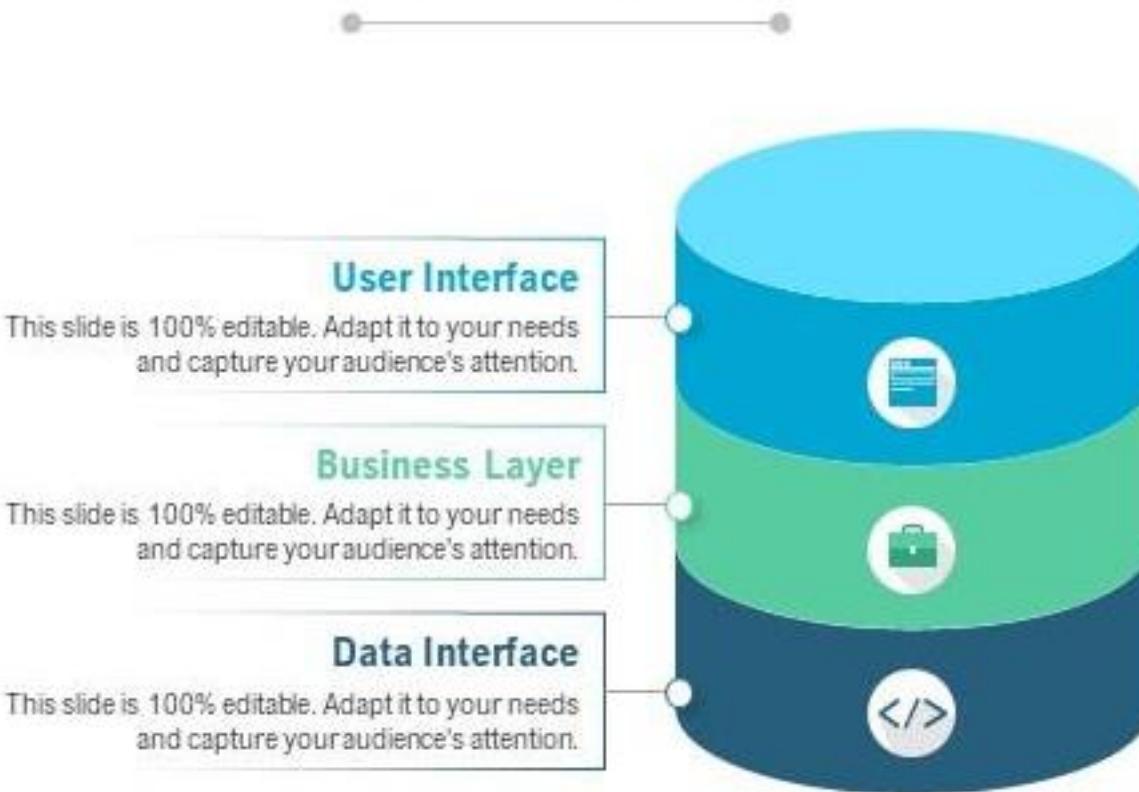
Problemas de rendimiento

Baja confiabilidad (Degradación)

One Stack (Una sola tecnología)

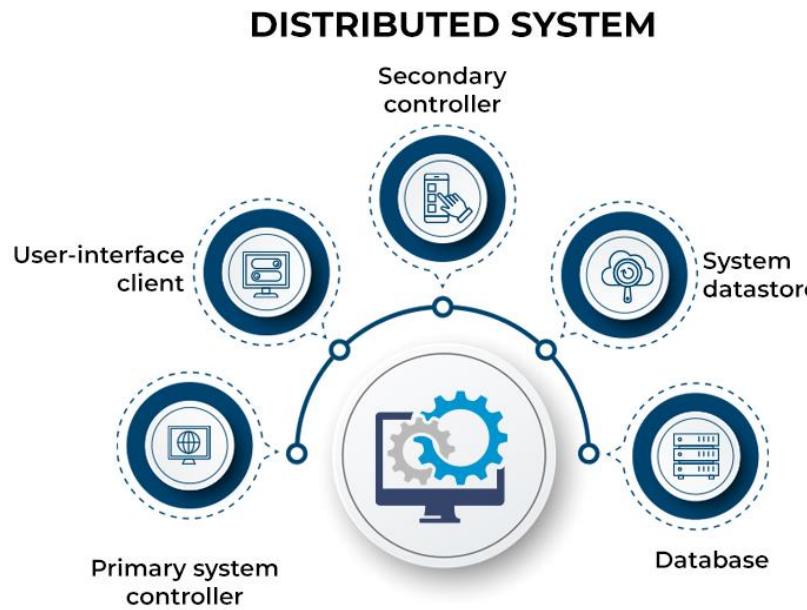
# Demo simplificada

## MONOLITHIC ARCHITECTURE



# Aplicaciones SOA

## Sistema Distribuido



Un sistema distribuido es una aplicación de software en la que los componentes están ubicados en computadoras en red y se comunican y coordinan sus acciones emitiendo llamadas o pasando mensajes

# → Introducción a Microservicios

## Service-oriented Architecture



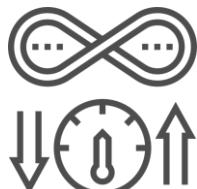
## Falacias de la computación distribuida



La red es confiable



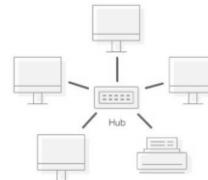
La latencia es cero



El ancho de banda es infinito



La red es segura



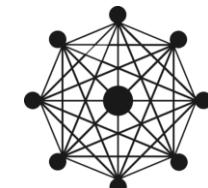
La topología no cambiará



Hay un administrador



El costo de transporte es cero



La red es homogénea.

# → Introducción a Microservicios

## Acoplamiento

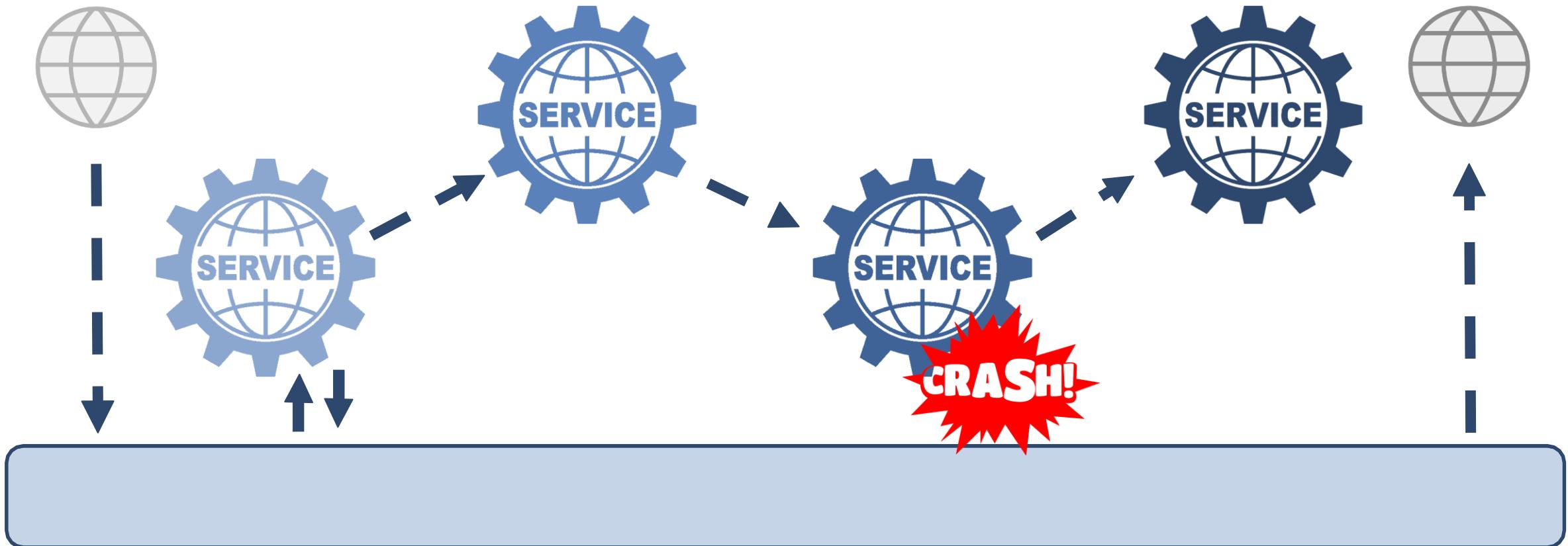
**Plataforma**  
(Asociado a una misma  
tecnología)

**Comportamiento**  
(Conocimiento de la firma)

**Temporal**  
(Dependencia entre  
servicios)

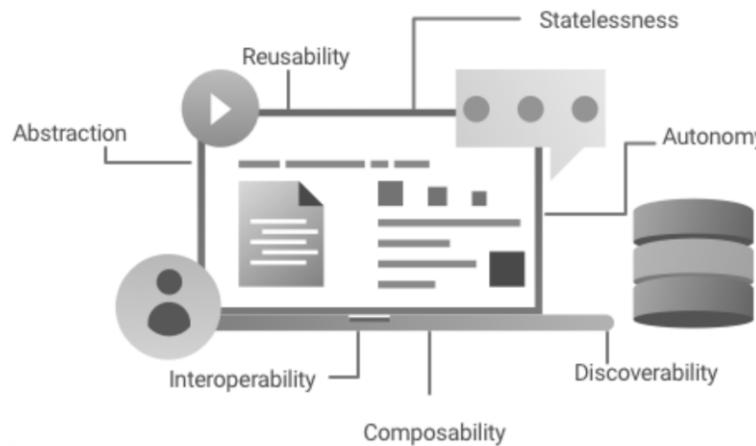


## Remote Procedure Call



# → Introducción a Microservicios

## ¿Qué es Arquitectura SOA?



Modelo/Estilo de arquitectura fundamentado en el paradigma de diseño de la **orientación a servicios**.

Evolución de la industria en metodologías, tecnología y estándares.

Nueva generación de la **computación distribuida**.

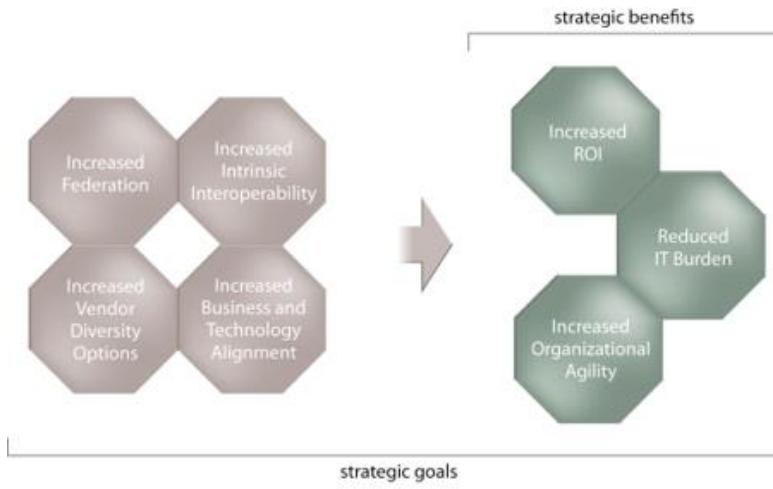
Brinda beneficios estratégicos a las organizaciones.

Define los siguientes elementos :

- Servicios
- Contratos
- Mensajes

# → Introducción a Microservicios

## ¿Qué es Arquitectura SOA?

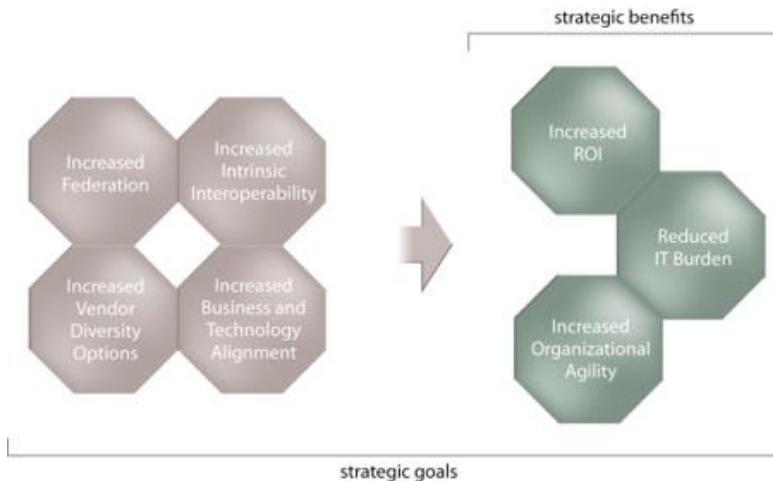


Objetivos y beneficios estratégicos.

- Incrementar la interoperabilidad intrínseca.
- Incrementar la alineación de TI con el Negocio.
- Agilidad Organizacional.
- Incrementar el ROI

# → Introducción a Microservicios

## ¿Qué es Arquitectura SOA?

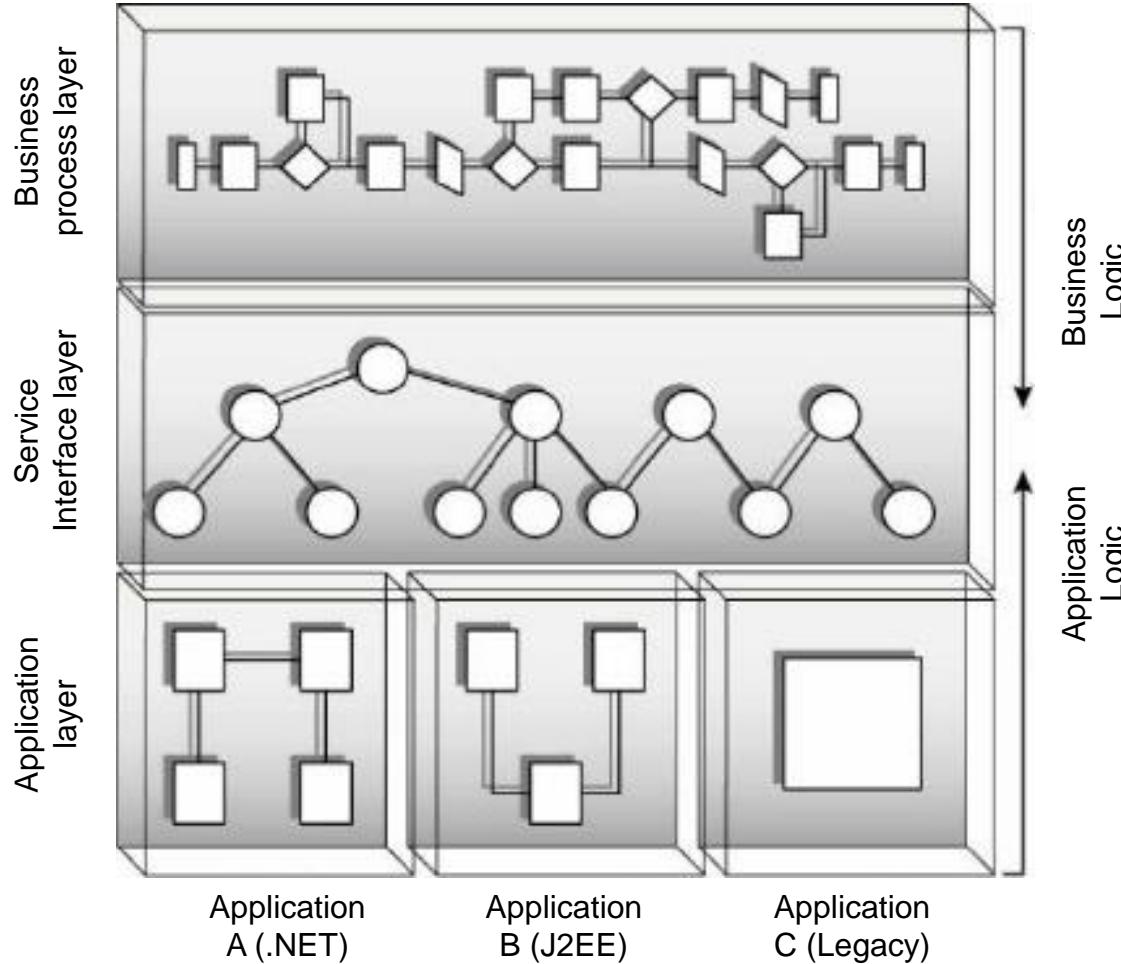


"SOA: Principles of Service Design", Copyright Prentice Hall/PearsonPTR

- Todos los objetivos son estratégicos en naturaleza, brindando beneficios a largo plazo.
- En comparación con objetivos tácticos, los cuales se basan en requerimientos inmediatos a corto plazo.
- Una característica distintiva con la computación orientada a servicios es su naturaleza estratégica.
- Contrario a la naturaleza táctica del desarrollo de aplicaciones basadas en silos.

# → Introducción a Microservicios

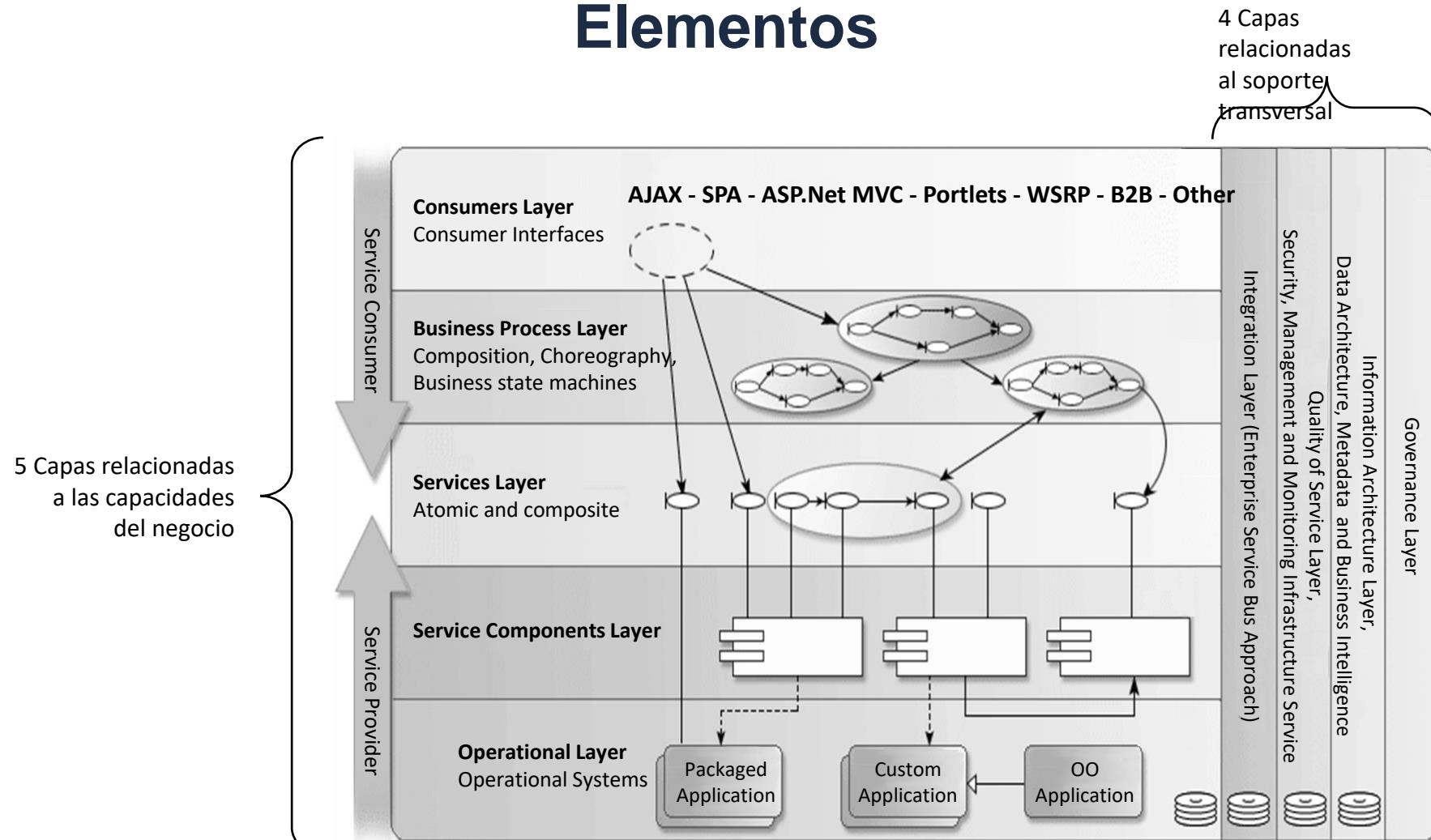
## Elementos



- |   |   |
|---|---|
| <b>Procesos</b> <ul style="list-style-type: none"> <li>• Personas</li> <li>• Reglas</li> <li>• Lógicas</li> <li>• Mensajes</li> </ul>   | <b>Orchestration service layer</b><br><b>Task-Centric Business services</b><br><b>Entity-Centric Business services</b><br><b>Util Application services</b> <ul style="list-style-type: none"> <li>• Contratos</li> <li>• Lógicas</li> <li>• Mensajes</li> </ul> |
| <b>Recursos de TI</b> <ul style="list-style-type: none"> <li>• Legados</li> <li>• Base de datos</li> <li>• Componentes</li> <li>• Frameworks</li> <li>• Redes</li> <li>• Otros</li> </ul> |   |

# Introducción a Microservicios

## Elementos



# → Introducción a Microservicios

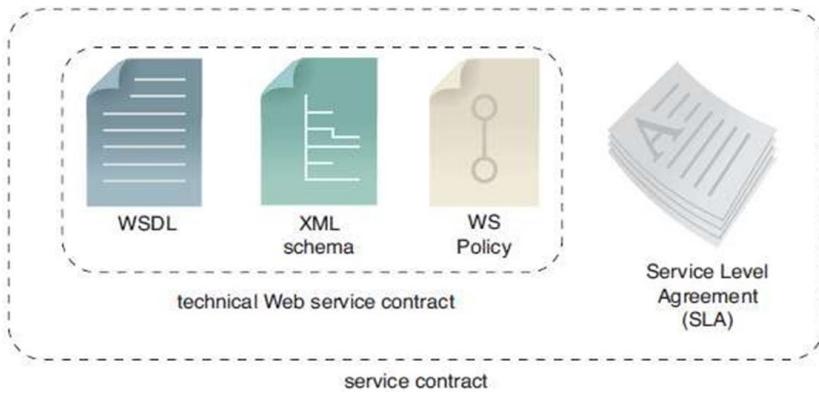
## Elementos - Servicio



- Unidad fundamental de SOA
- Un contenedor de una o muchas capacidades
- Diseñado con capacidades reutilizables
- Servicios como: Web Services, Components, REST Services.

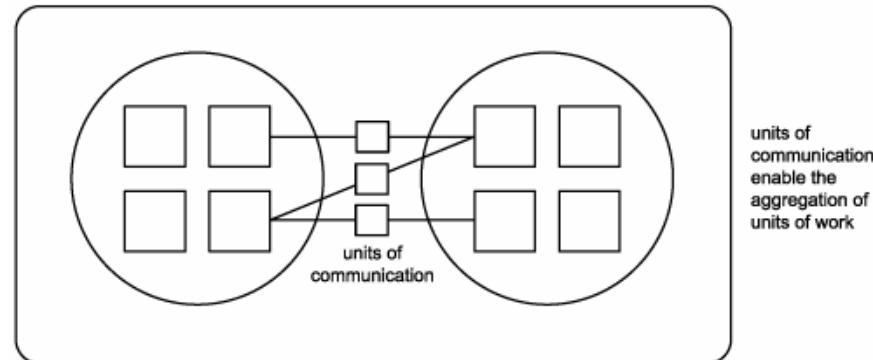
# → Introducción a Microservicios

## Elementos – Contratos



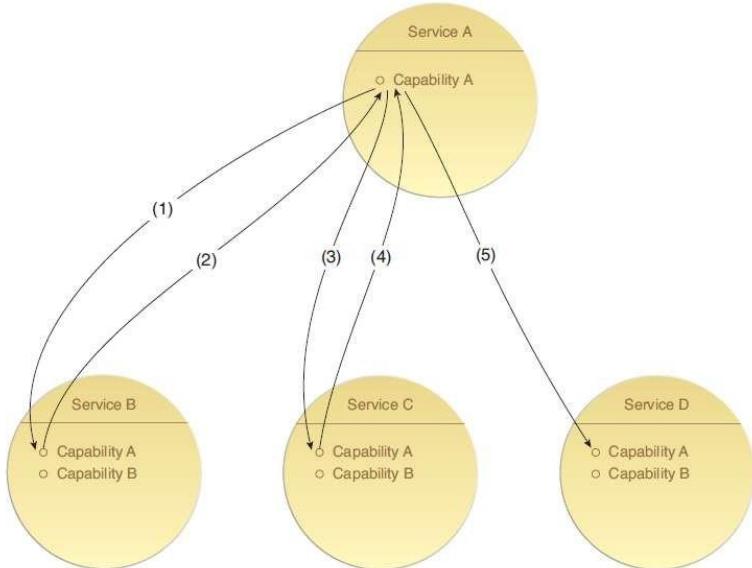
- El contrato expresa información acerca del servicio, sus capacidades y **tipos de datos**.
- Un servicio consumidor debe cumplir con los requerimientos expuestos en el contrato.
- La parte fundamental del contrato del servicio es su **interface técnica**.
- El contrato del servicio se puede complementar con un documento de SLA que describa información adicional de características y limitaciones.
- El como se diseñen y existan **físicamente** los servicios depende sobre la tecnología que se utiliza para crear el servicio.
- El principio de **estandarización del contrato** es dedicado a la definición del contrato, aspecto muy importante para lograr beneficios estratégicos.

## Elementos – Mensajes



- Unidad de comunicación de los servicios.
- La estandarización de los mensajes permite:
  - La interoperabilidad intrínseca
  - Reducir costos de mantenimiento
  - Eliminar el uso de transformaciones que impactan el rendimiento

## Composición



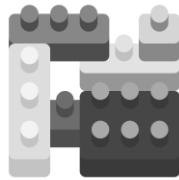
- Agregación coordinada de servicios
- Reutilización de capacidades para diferentes procesos de negocio

# Introducción a Microservicios

## Principios de Diseño



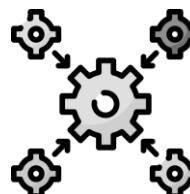
Estandarización del contrato



Bajo acoplamiento del servicio



Abstracción del Servicio



Capacidad del Servicio de ser Reutilizado



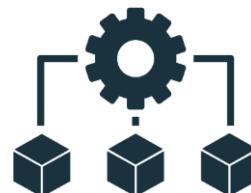
Autonomía del Servicio



Servicio sin Estado



Capacidad del Servicio de ser Descubierto e interpretado



Capacidad del Servicio de ser Compuesto

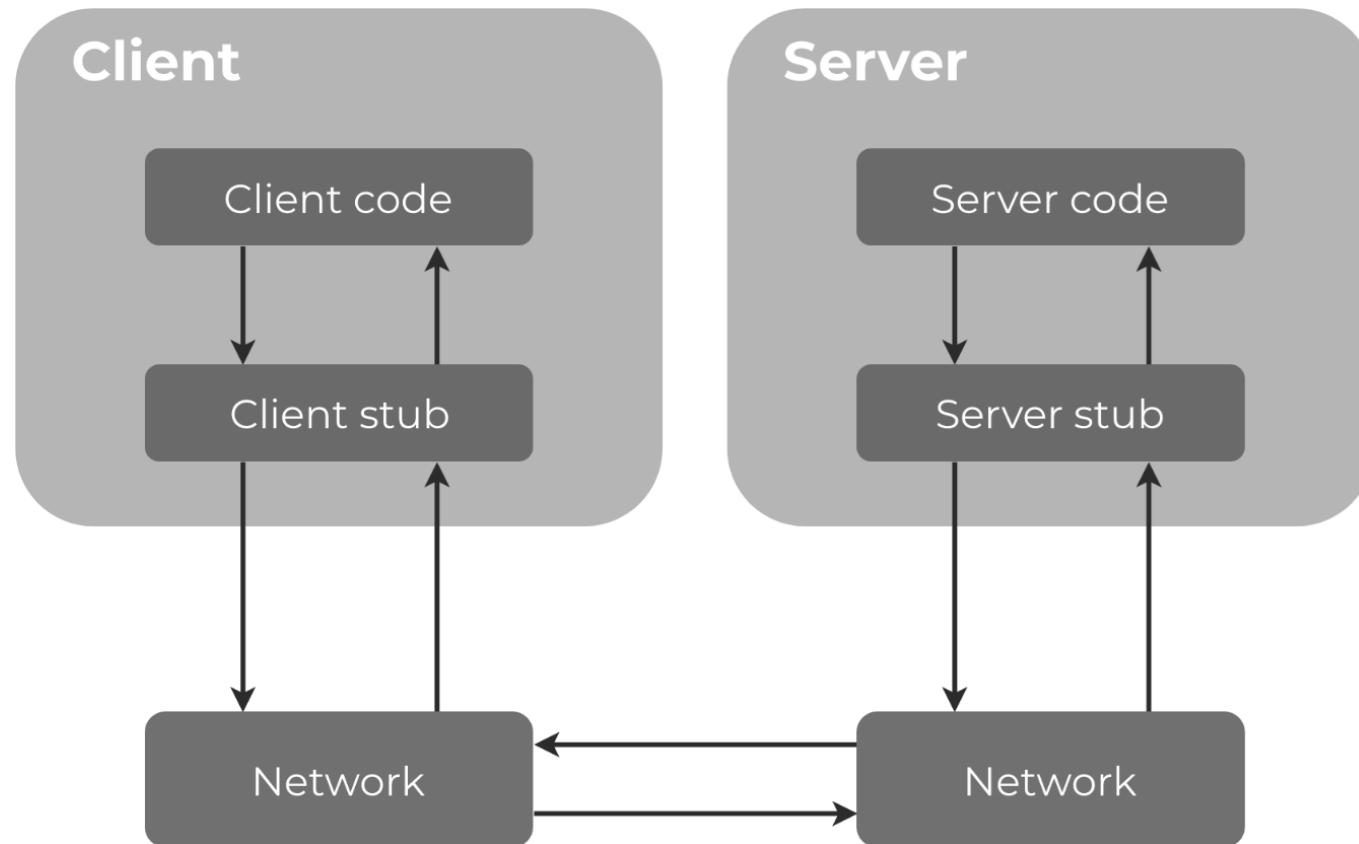
# → Introducción a Microservicios

## RPC, REST



# → Introducción a Microservicios

## ¿Qué es RPC?



## Remote Procedure Call

Llamar a un método de manera remota  
.NET Remoting/Java RMI

Acoplamiento a nivel de  
plataforma



Acoplamiento a nivel de  
comportamiento



Acoplamiento a nivel  
temporal



# → Introducción a Microservicios

## Remote Procedure Call: SOAP

Simple Object Access Protocol  
(SOAP)

Call method using standarized XML  
(Web Services Description Language  
WSDL)

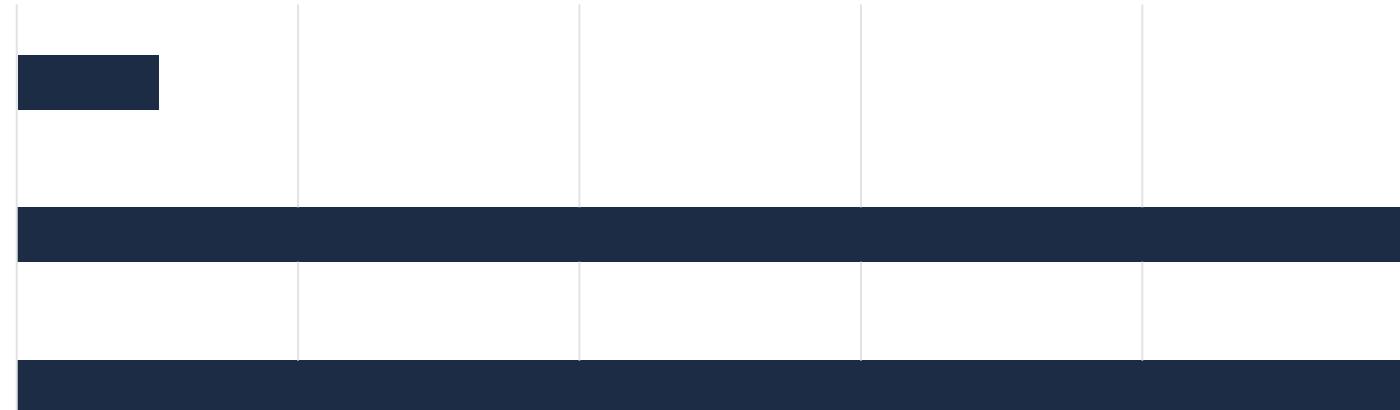
Acoplamiento a nivel de  
plataforma



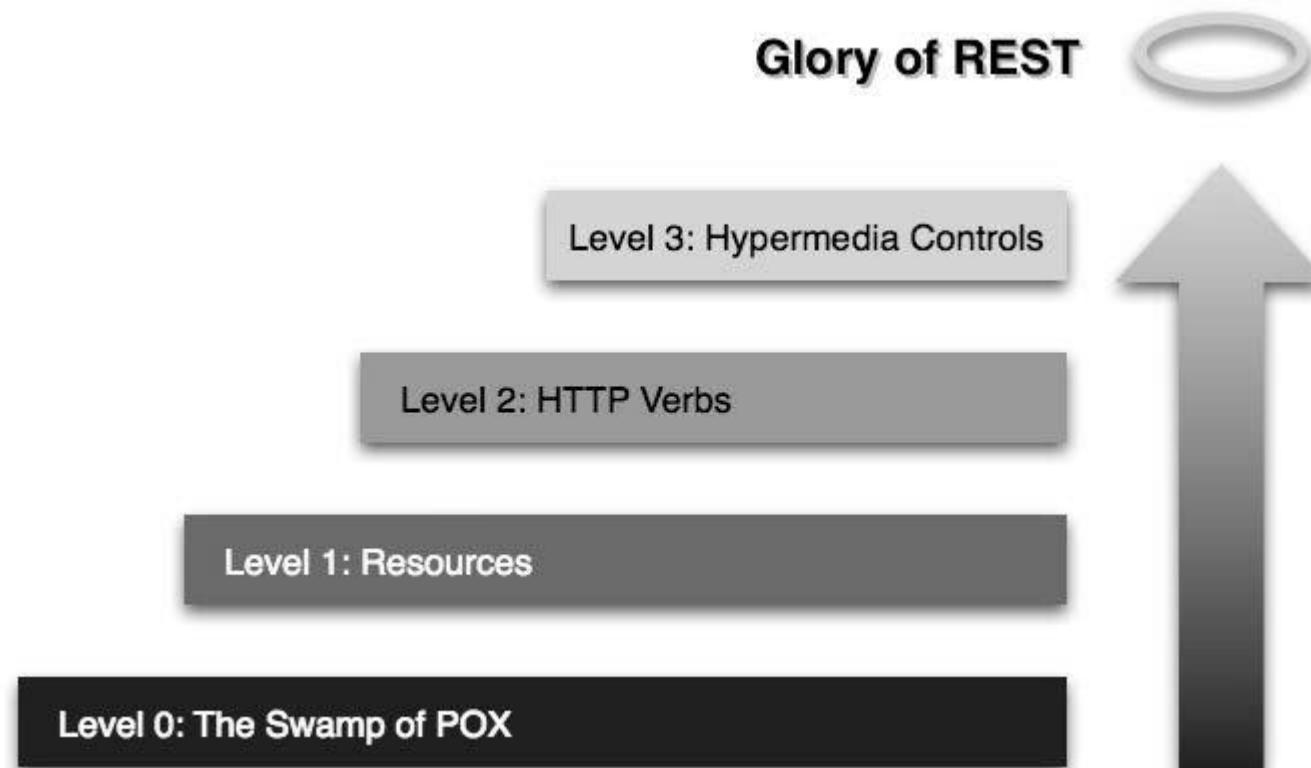
Acoplamiento a nivel de  
comportamiento



Acoplamiento a nivel  
temporal



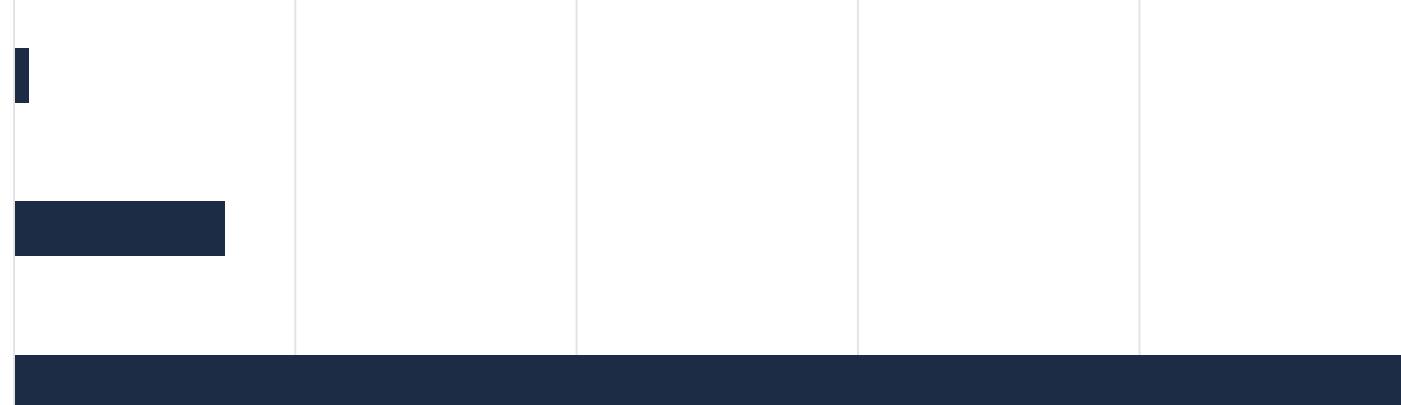
## Representational State Transfer (REST)



# → Introducción a Microservicios

## Representational State Transfer (REST)

Acoplamiento a nivel de plataforma



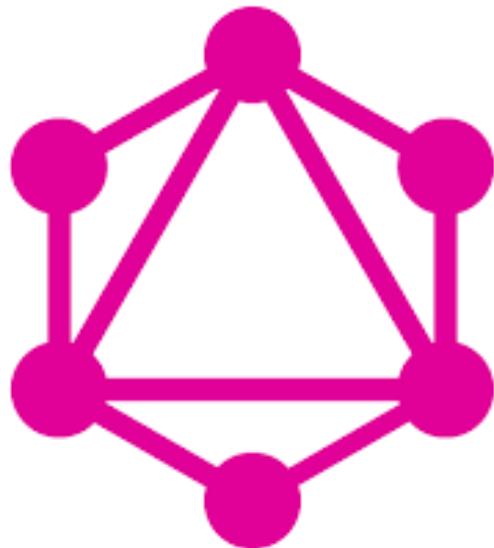
Acoplamiento a nivel de comportamiento

Acoplamiento a nivel temporal

# → Introducción a Microservicios

## GraphQL y gRPC.

## GraphQL

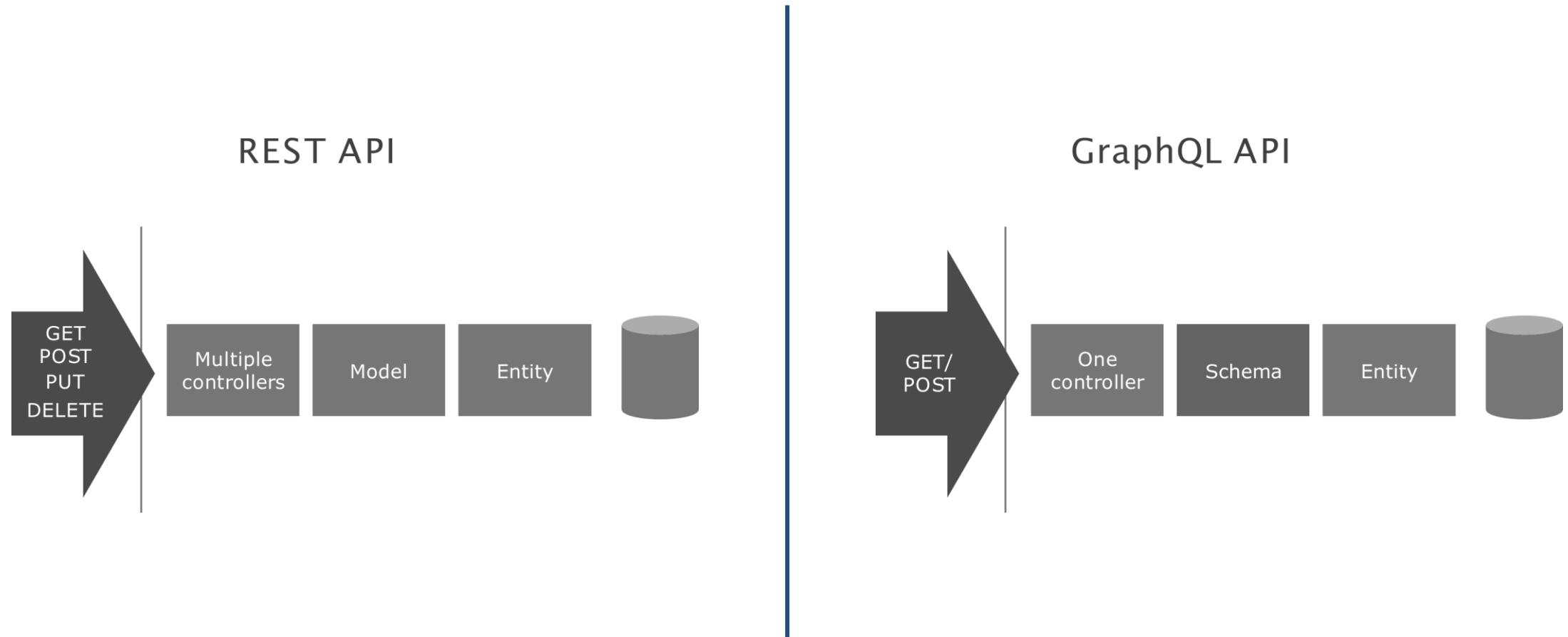


GraphQL es un lenguaje de consulta para LAS API y un tiempo de ejecución para satisfacer esas consultas con los datos existentes. GraphQL proporciona una descripción completa y comprensible de los datos de la API, ofrece a los clientes el poder de pedir exactamente lo que necesitan y nada más, facilita la evolución de las API con el tiempo y permite potentes herramientas de desarrollo.

<https://graphql.org/>

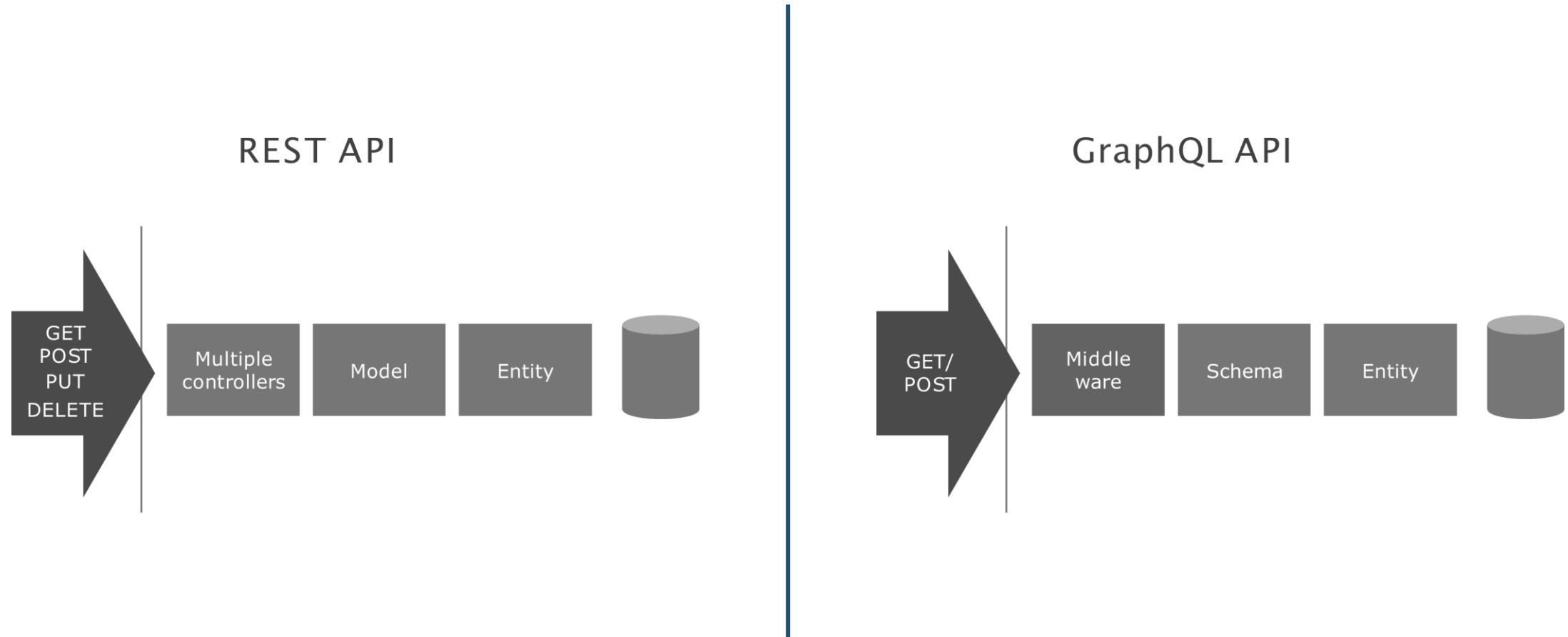
# → Introducción a Microservicios

## GraphQL



# → Introducción a Microservicios

## GraphQL



# → Introducción a Microservicios

## Diferencias entre GraphQL y REST

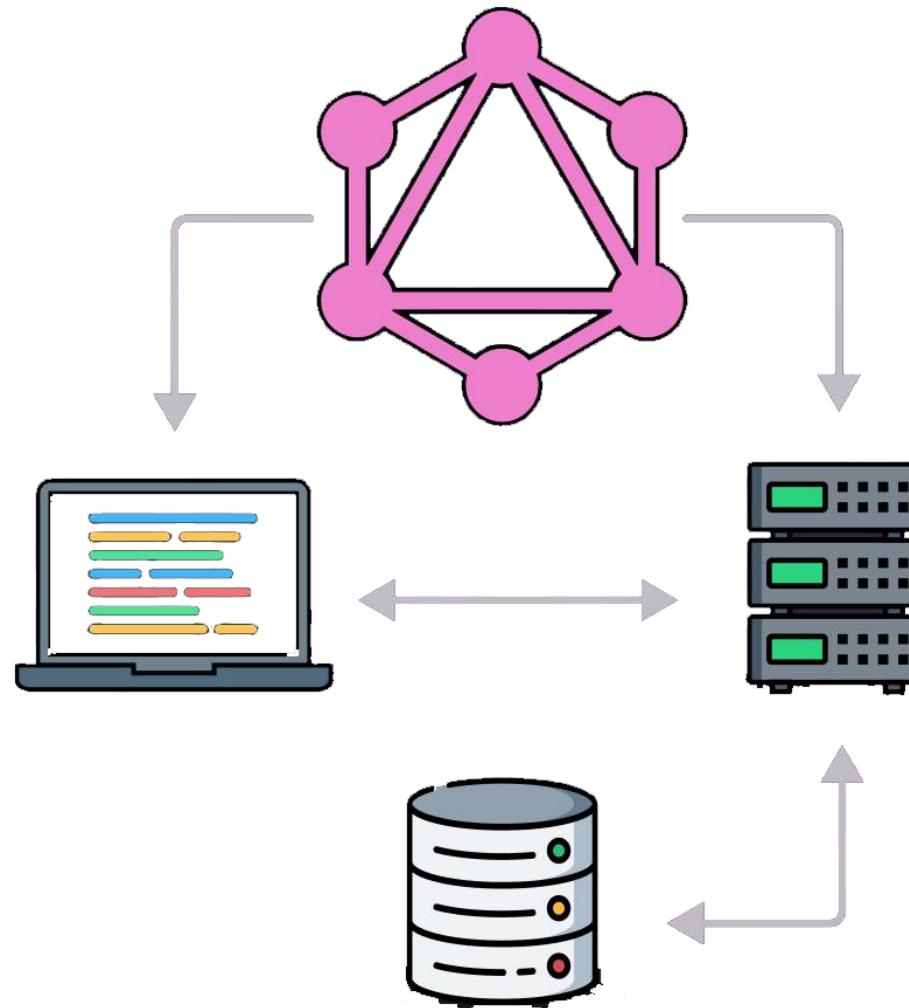
### REST

- **Es solo una convención:** Es una manera de comunicarse entre el servidor y cliente, cada uno tiene sus reglas.
- **El servidor expone recursos:** Los clientes se tienen que adecuarse a como están expuestos.
- **Hace overfetching:** Envía más información de la que se necesita.
- **Múltiples request por vista:** Muy costoso en performance, básicamente es una aplicación en blanco que aún no ha cargado datos o tiene custom endpoints.
- **Documentación ajena al desarrollo:** No hay un estándar por lo que depende mucho del desarrollador para mantenerla.

### GraphQL

- **Lenguaje tipado y validable:** Le damos una forma de lo que recibe y lo que devolvemos, Además de agregarle seguridad.
- **El Cliente define que recibe:** Haciendo una consulta, de la estructura que se define como respuesta.
- **Envía lo necesario:** Se tiene control total de las respuestas que se esperan del servidor.
- **Hace un solo request por vista:** Se maneja un solo row, prácticamente en solo request puedes mandar todo lo que necesitas.

# Demo GraphQL



# → Introducción a Microservicios

## gRPC



<https://grpc.io/docs/what-is-grpc/core-concepts/>  
<https://developers.google.com/protocol-buffers>

Al igual que muchos sistemas RPC, gRPC se basa en la idea de definir un servicio, especificando los métodos que se pueden llamar de forma remota con sus parámetros y tipos de valor devuelto.

De forma predeterminada, gRPC utiliza Protocol Buffers como el lenguaje de definición de interfaz (IDL) para describir tanto la interfaz de servicio como la estructura de los mensajes de carga útil. Es posible utilizar otras alternativas si se desea.

## gRPC - Ventajas



- Marco de RPC moderno, ligero y de alto rendimiento.
- Desarrollo de la API de primer contrato utilizando búferes de protocolo de forma predeterminada, lo que permite realizar implementaciones independientes del idioma.
- Dispone de herramientas para muchos idioma con la finalidad de generar clientes y servidores fuertemente tipados.
- Admite llamadas de transmisión en secuencias bidireccionales, de servidor y de cliente.
- Uso reducido de red con serialización binaria Protobuf.

DEMO

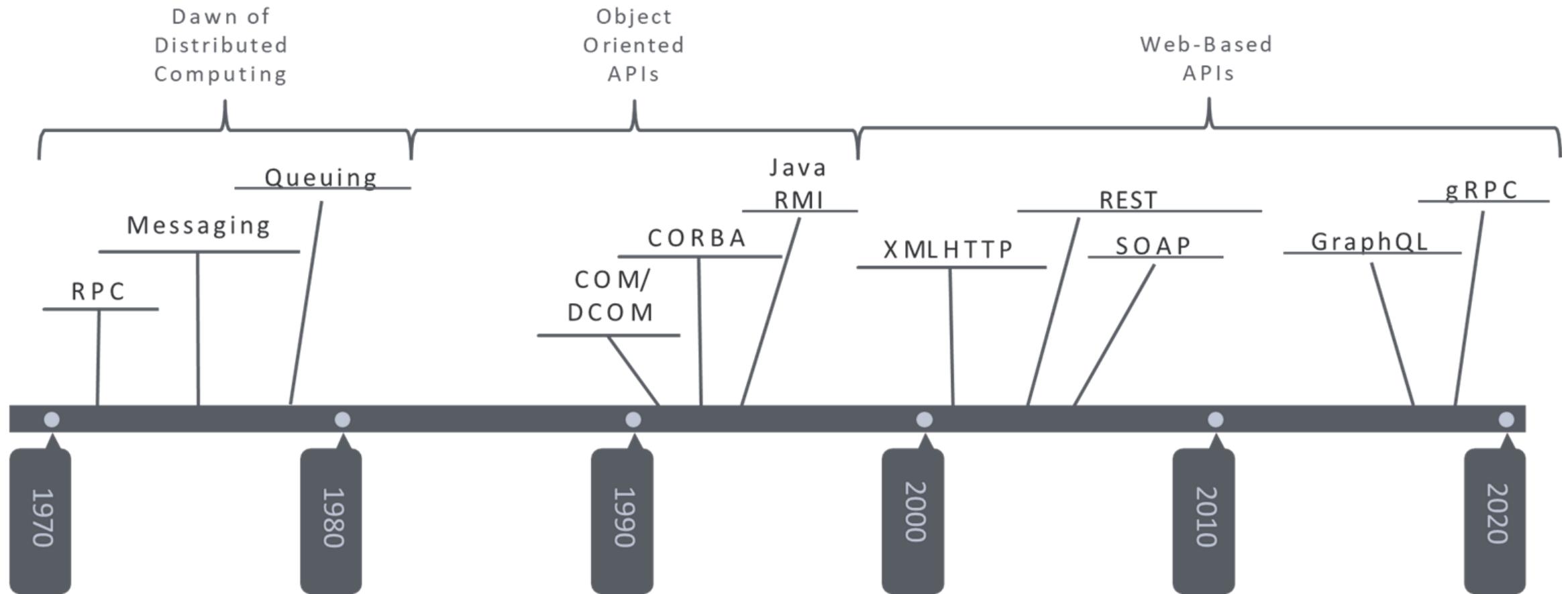


# Demo gRPC



# → Introducción a Microservicios

## Historia



# ¿Qué son microservicios?

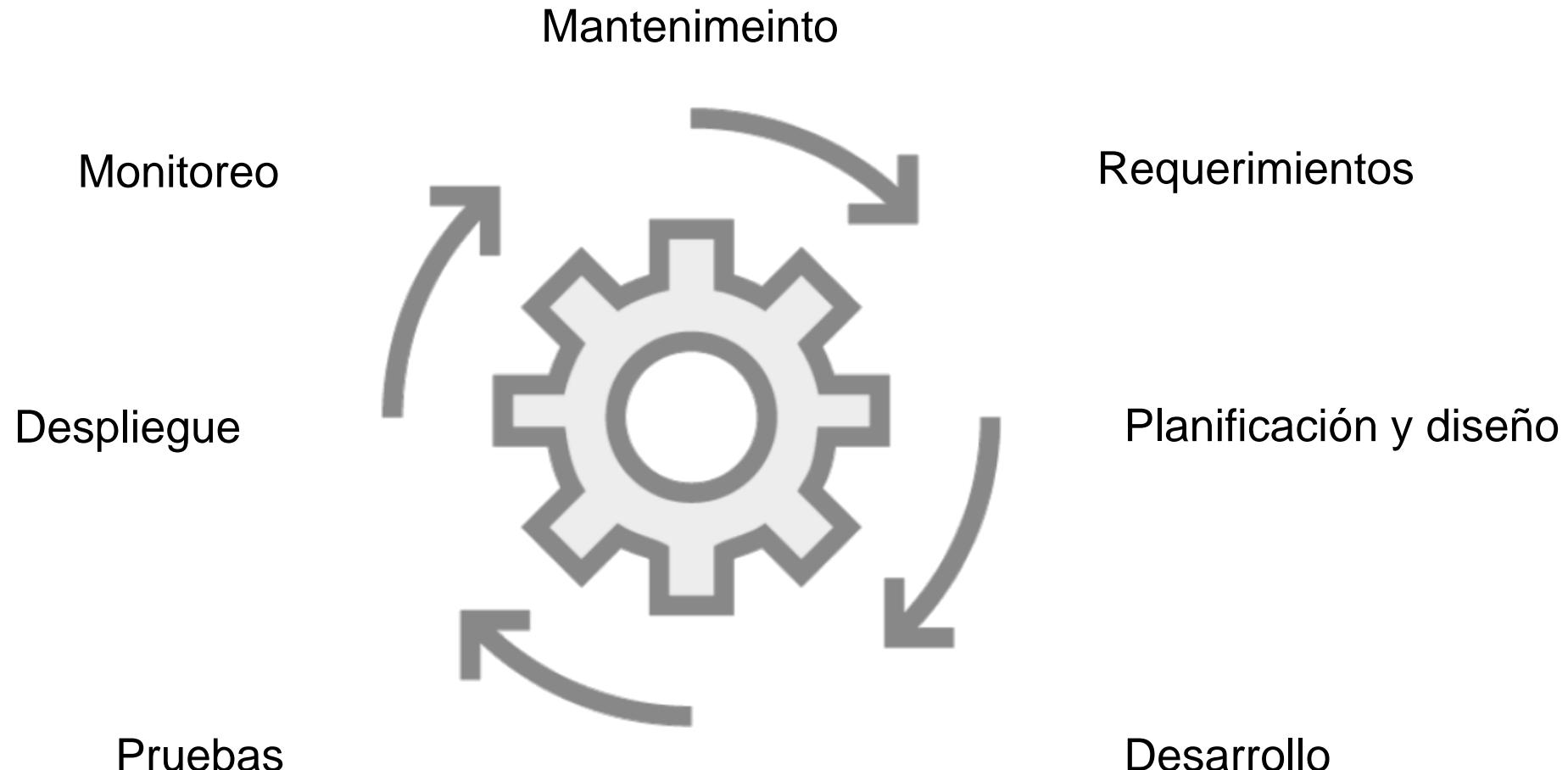
## Ciclo de vida de desarrollo de software



- Ciclo de vida del desarrollo de software
- Del proyecto al producto
- Organización y gestión.
- Desplegado en producción
- Los microservicios tienen un impacto en este ciclo de vida

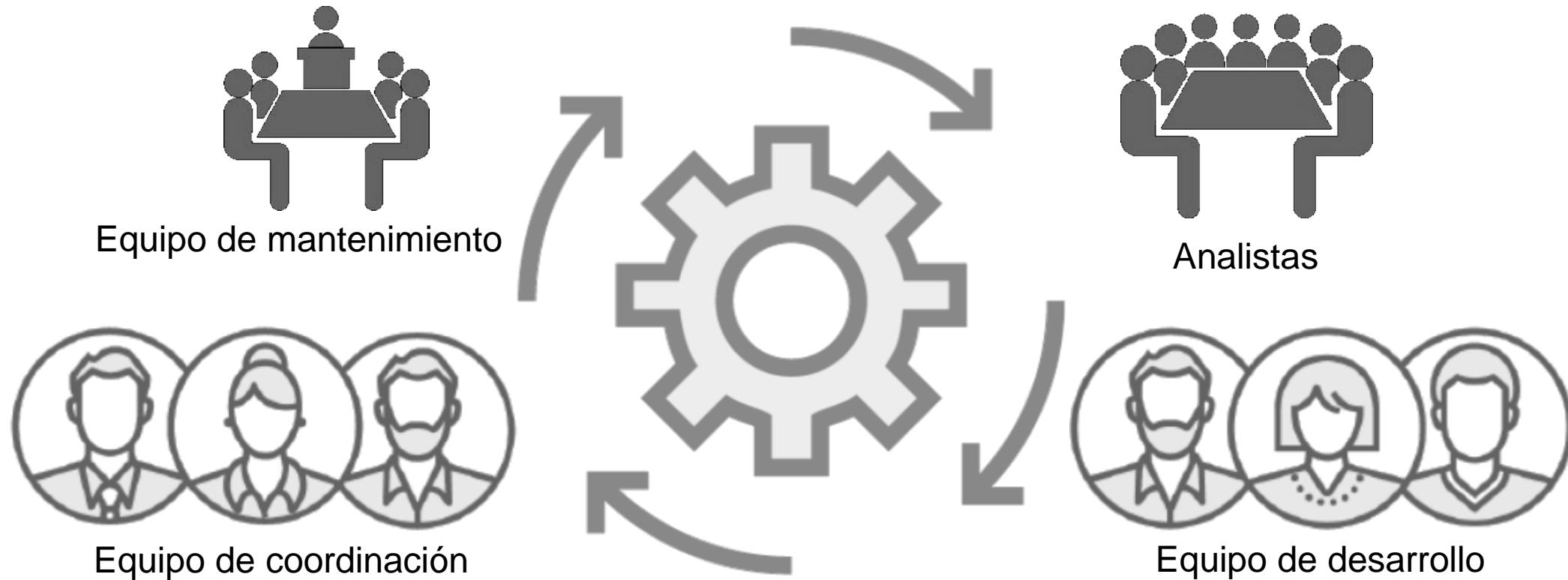
# → Introducción a Microservicios

## Ciclo de vida de desarrollo de software

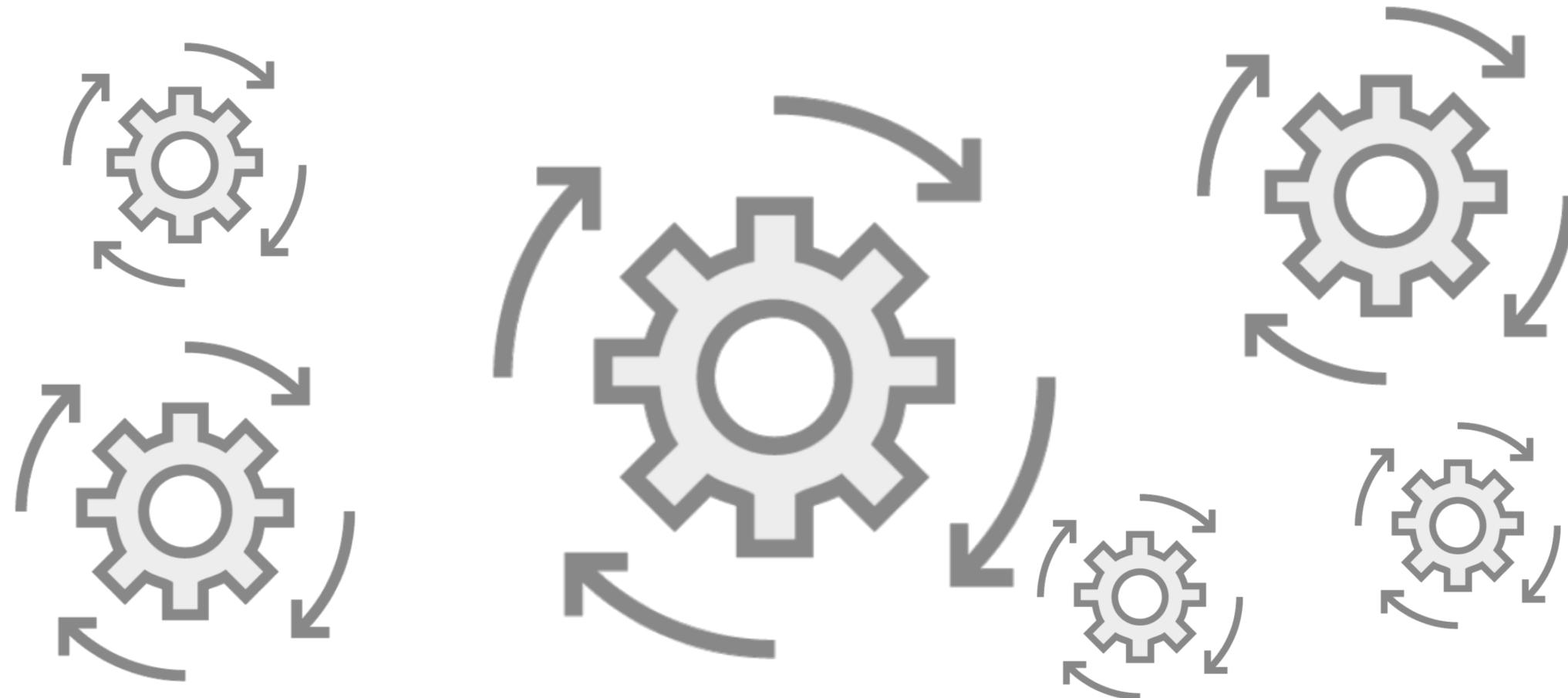


# → Introducción a Microservicios

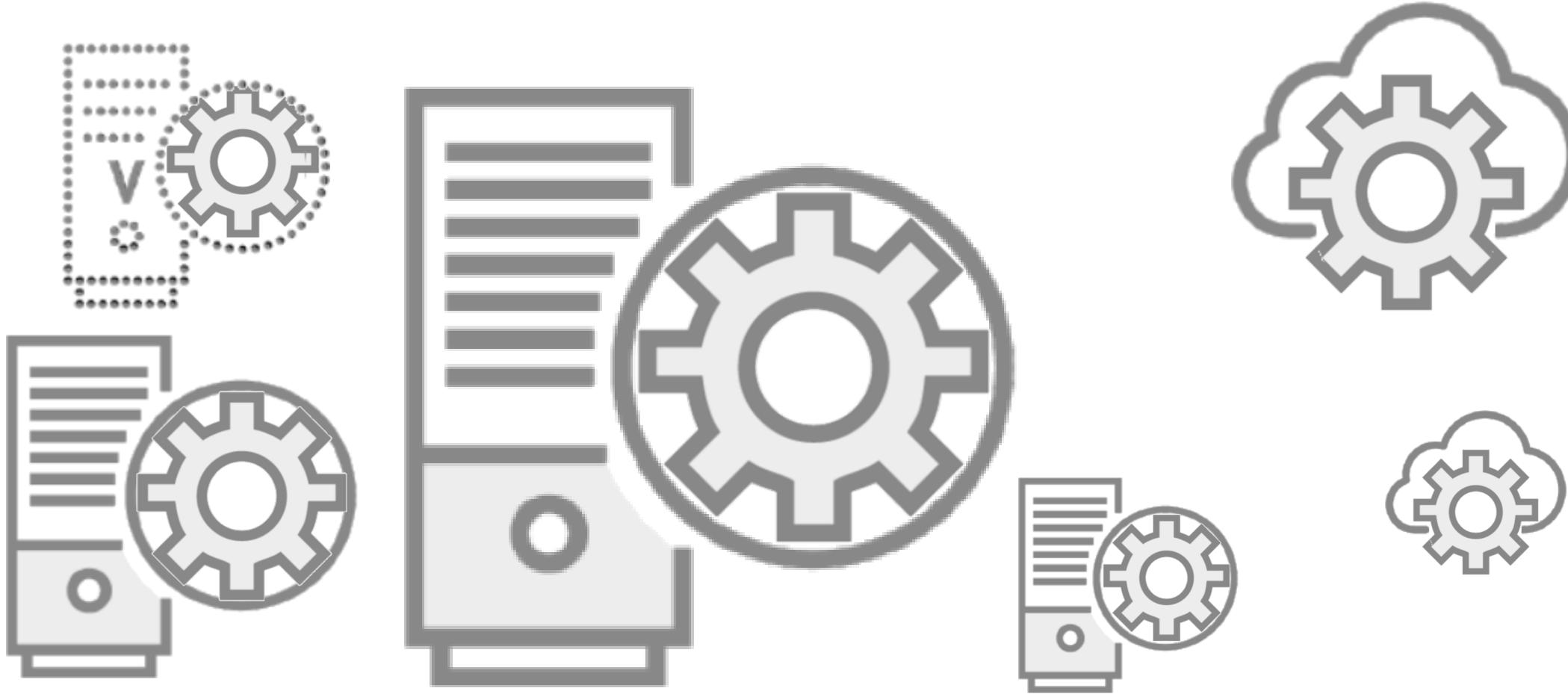
## Ciclo de vida de desarrollo de software



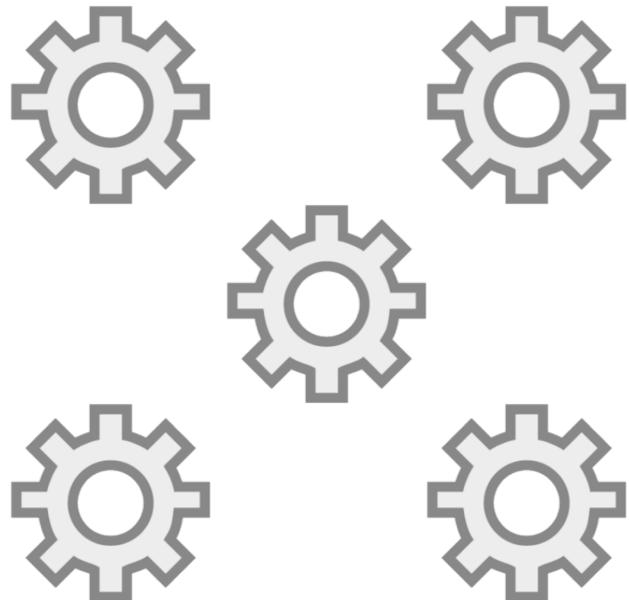
## Ciclo de vida de desarrollo de software



## Ciclo de vida de desarrollo de software



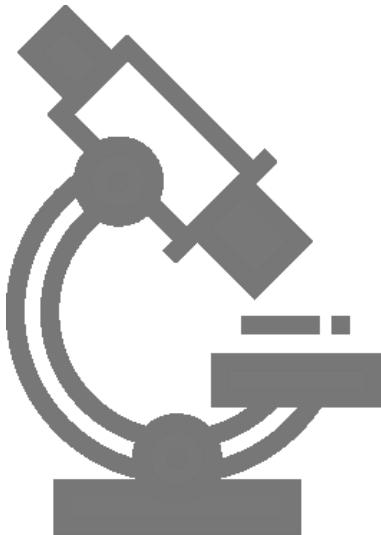
## Microservicios



- Conjunto de prácticas
- Aumentar la velocidad
- Escala
- Tecnología agnóstica
- Principios y patrones arquitectónicos.

# → Introducción a Microservicios

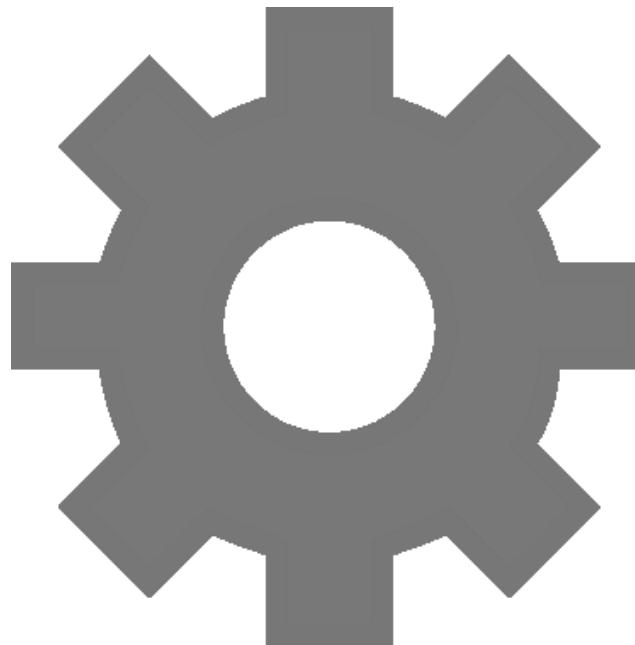
## Micro



- Grande o pequeño
- Ninguna medida universal
- "Hace una cosa"
- Alcance de funcionalidades
- Contexto limitado
- Identificar subdominios

# → Introducción a Microservicios

## Servicio



- Componente desplegable independientemente
- Interoperabilidad
- Comunicación basada en mensajes
- Arquitectura orientada a servicios (SOA)

# → Introducción a Microservicios

## Microservicio (I)

El estilo arquitectónico de microservicios es un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno de los cuales se ejecuta en su propio proceso y se comunica con mecanismos ligeros.

James Lewis and Martin Fowler, Thoughtworks



# → Introducción a Microservicios

## Microservicio (II)

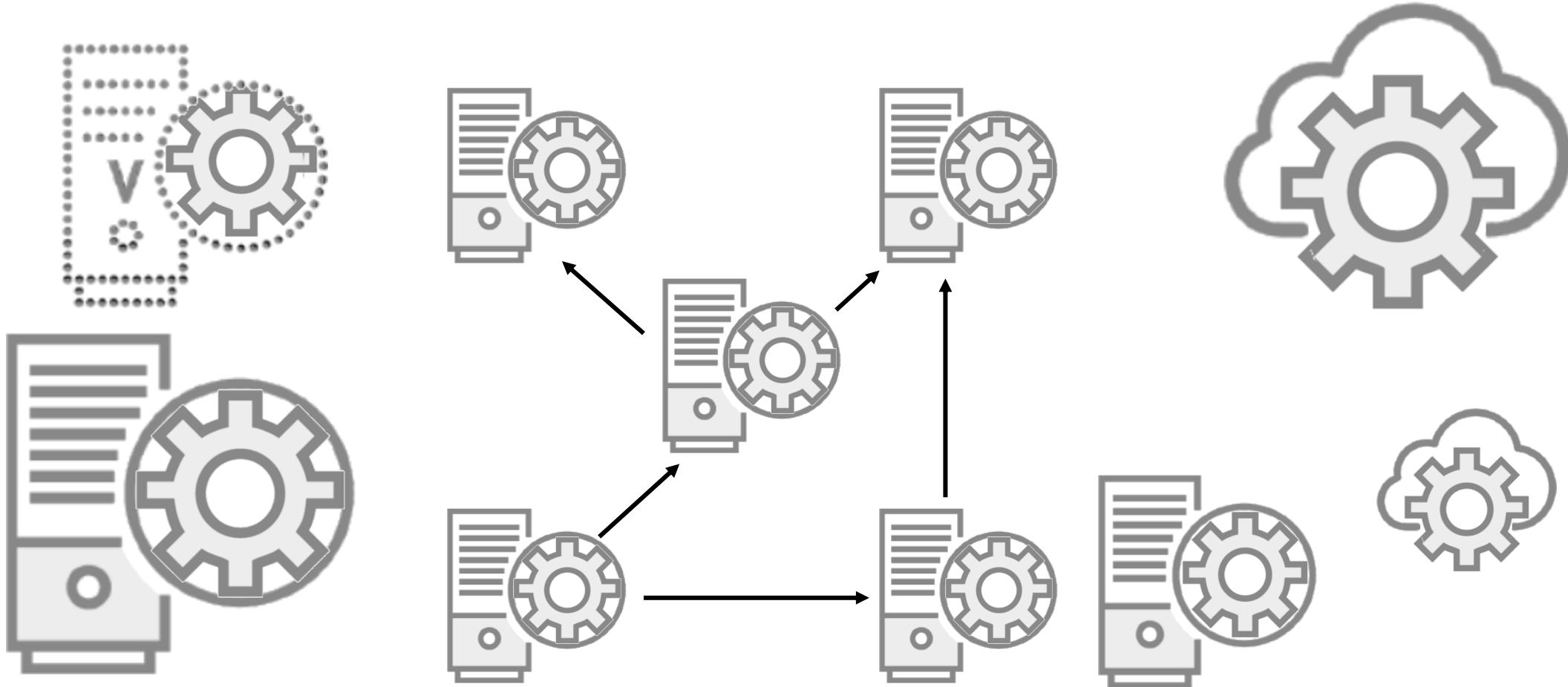
Un estilo de arquitectura de software, en el que las aplicaciones complejas se componen de pequeños procesos autónomos que se comunican entre sí mediante API independientes del lenguaje.



## Microservicios

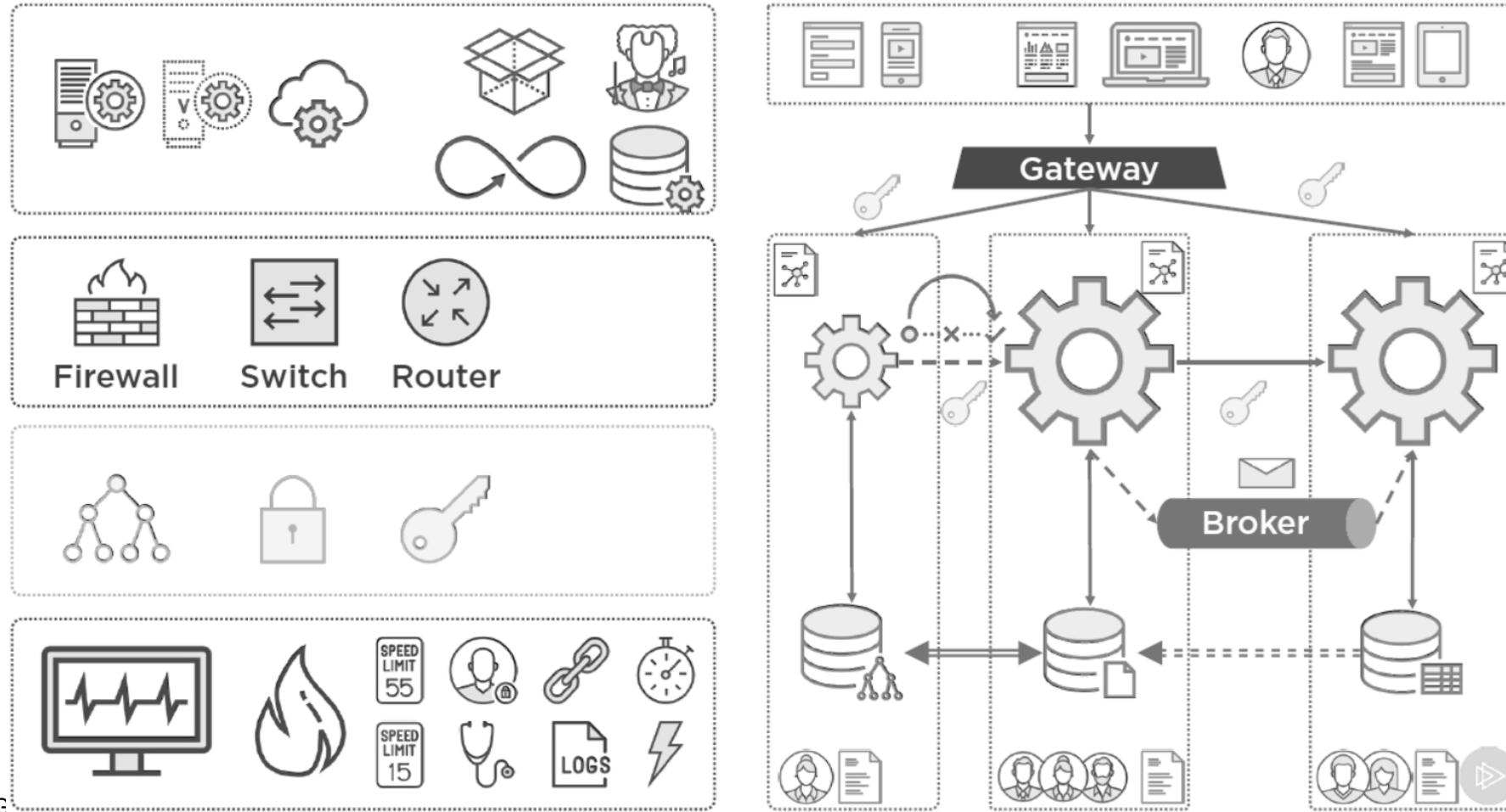


## Microservicios



# → Introducción a Microservicios

## Elementos de un microservicio

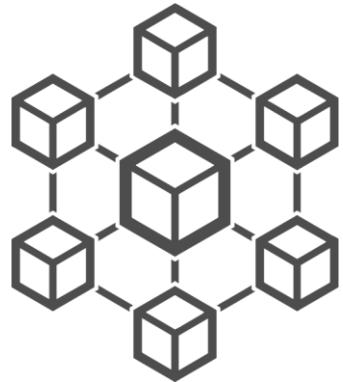


# → Introducción a Microservicios

¿Son los microservicios adecuados para mi organización?



## Beneficios de los microservicios



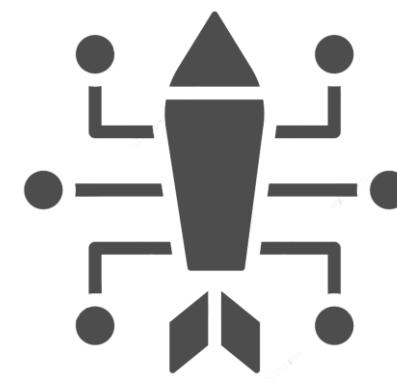
### Pequeños servicios

Puede ser propiedad de un equipo  
Más fácil de entender  
Puede ser reescrito



### Elección de tecnología

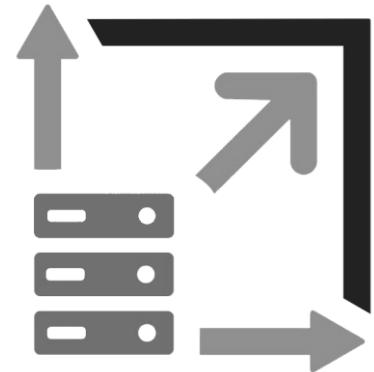
Adopta nueva tecnología  
Usa la herramienta adecuada  
Estandarizar donde tiene sentido



### Despliegue individual

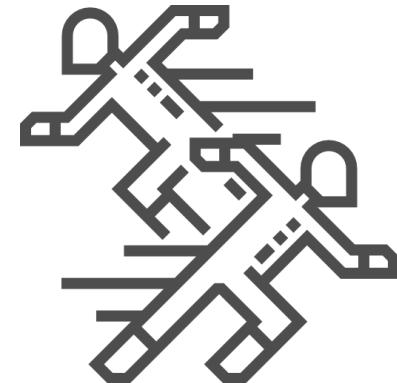
Menor riesgo  
Minimiza el tiempo de inactividad  
Actualizaciones frecuentes

## Beneficios de los microservicios



### Escalabilidad

Escalar servicios  
individualmente  
Económico



### Agilidad

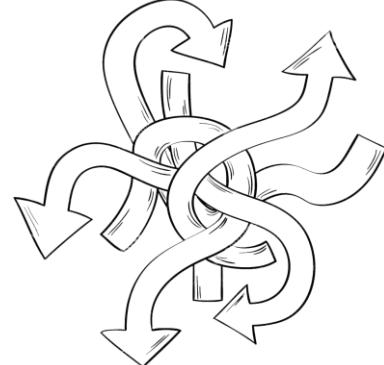
Adaptarse rápidamente  
Reutilización más fácil

## Desafíos de los microservicios



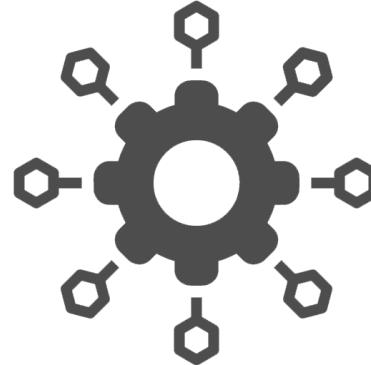
### Productividad del desarrollador

¿Cómo podemos facilitar que los desarrolladores sean productivos trabajando en el sistema?



### Interacciones complejas

Tenga cuidado realizando comunicaciones ineficientes e innecesarias entre microservicios



### Despliegue

Necesitarás automatizar el proceso



### Monitoreo

Necesitamos un lugar centralizado para verificar los registros y monitorear los problemas

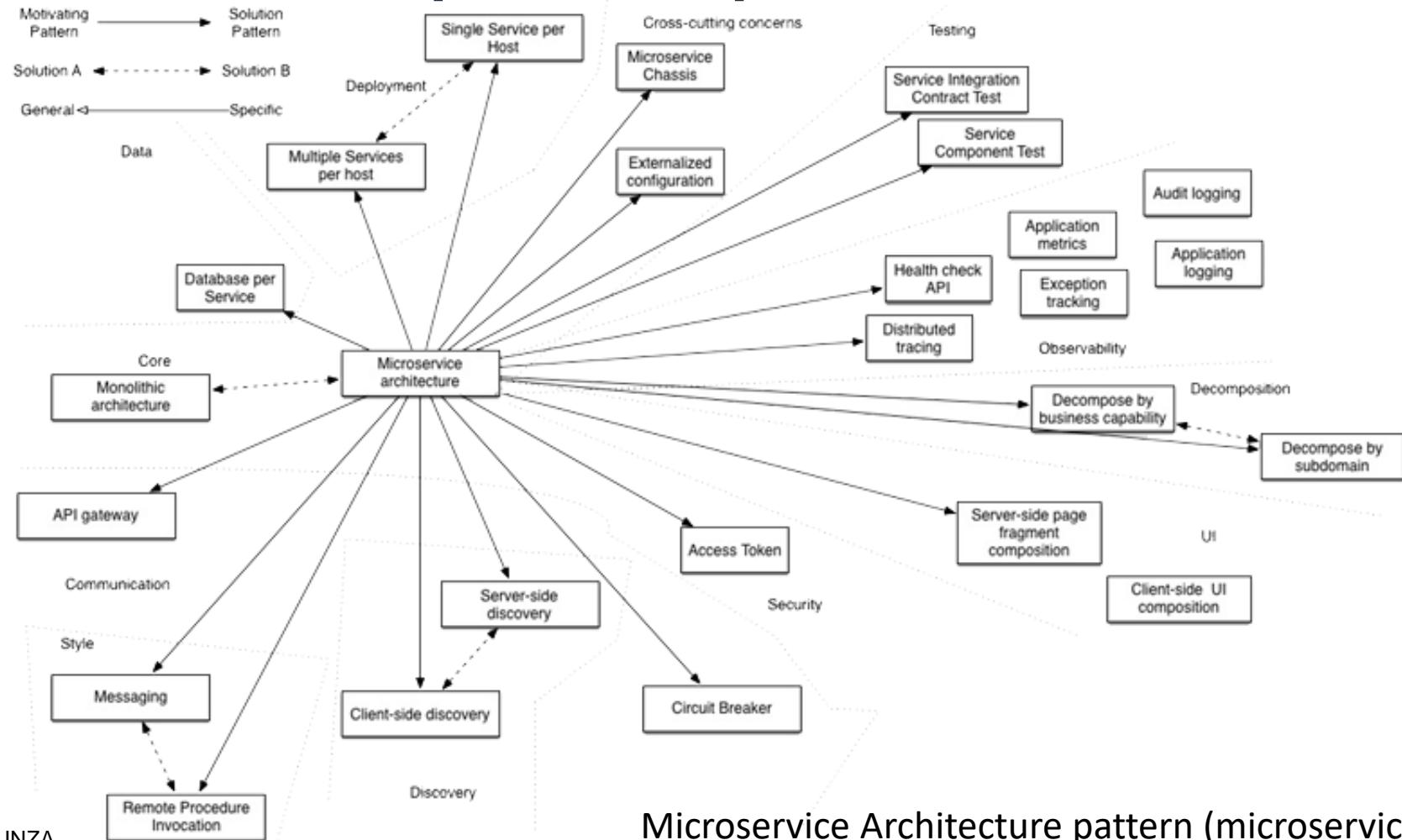
02



# Principales patrones de microservicios y su desarrollo en el curso

# → Principales patrones de microservicios

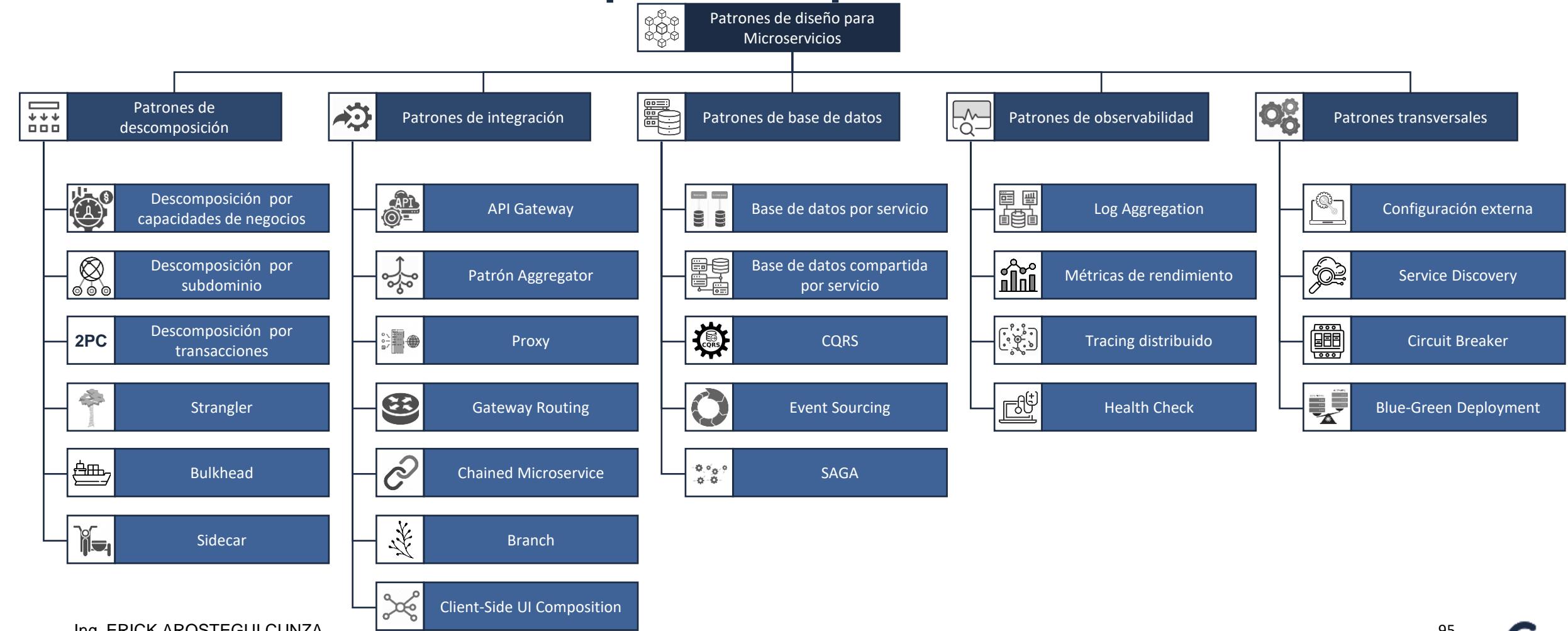
## Patrones de diseño para la arquitectura de microservicios



Microservice Architecture pattern (microservices.io)

# → Principales patrones de microservicios

## Patrones de diseño para la arquitectura de microservicios

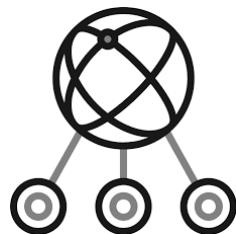


# → Principales patrones de microservicios

## Patrones de descomposición



Descomposición por capacidades de negocios



Descomposición por subdominio



Descomposición por transacciones



Strangler



Bulkhead



Sidecar

# → Principales patrones de microservicios

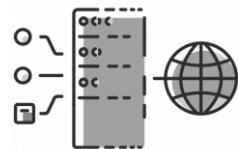
## Patrones de integración



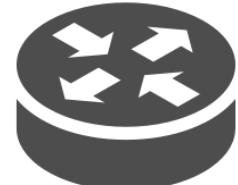
API Gateway



Patrón Aggregator



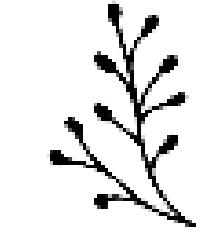
Proxy



Gateway Routing



Chained  
Microservice



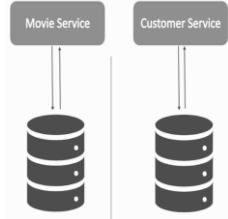
Branch



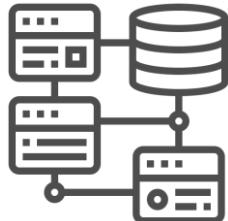
Client-Side UI  
Composition

# → Principales patrones de microservicios

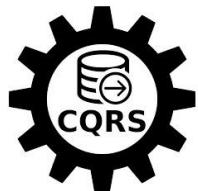
## Patrones de base de datos



Base de datos por servicio



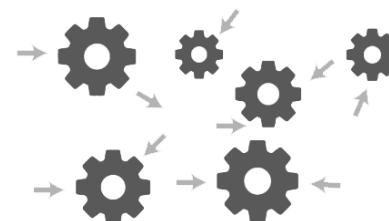
Base de datos compartida por servicio



CQRS



Event Sourcing



SAGA

# → Principales patrones de microservicios

## Patrones de observabilidad



Log Aggregation



Métricas de  
rendimiento



Tracing distribuido



Health Check

# → Principales patrones de microservicios

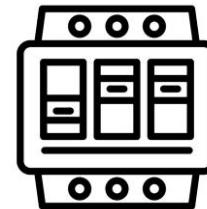
## Patrones transversales



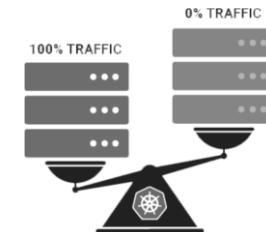
Configuración  
externa



Service Discovery



Circuit Breaker



Blue-Green  
Deployment

# → Principales patrones de microservicios

## Conclusiones

En el desarrollo de software empresarial a gran escala moderno, la arquitectura de microservicios puede ayudar a escalar el desarrollo con muchas ventajas a largo plazo. Pero **la arquitectura de microservicios no es una solución mágica** que se pueda usar en todos los casos de uso. **Si se utiliza en el tipo incorrecto de aplicación, la arquitectura de microservicios puede dar más dolores que ganancias.**

El equipo de desarrollo que desea adoptar la arquitectura de microservicios debe seguir un conjunto de procedimientos recomendados y usar un conjunto de patrones de diseño reutilizables y reforzados en batalla.



# → Principales patrones de microservicios

## Conclusiones

El patrón de diseño más importante en la arquitectura de microservicios es “**Database per Service**”. La implementación de este patrón de diseño es un desafío y necesita varios otros patrones de diseño estrechamente relacionados (**Event Sourcing, CQRS, Saga**).

En las aplicaciones empresariales típicas con varios clientes (Web, móvil, escritorio, Smart Devices), **las comunicaciones entre el cliente y los microservicios** pueden ser conversadores y pueden requerir control central con seguridad agregada. Los patrones de diseño **Backends For Front-end (BFF)** y **API Gateway** son muy útiles en estos escenarios.



# → Principales patrones de microservicios

## Conclusiones

Además, el patrón de **Circuit Breaker** puede ayudar en gran medida a **controlar los escenarios de error** en este tipo de aplicaciones.

**Migrar la aplicación monolítica heredada** a microservicios es bastante difícil, y el patrón de **Strangler** puede ayudar a la migración.

Al mismo tiempo, **Externalized Configuration** es un patrón obligatorio en cualquier desarrollo de aplicaciones modernas.



03



# Desarrollo de la arquitectura base de los microservicios, contenerización (Docker)

# **Virtualización (¿Qué es virtualización y tipos de virtualizaciones?).**

# → Contenerización

## Virtualizaciones

### Arquitectura de contenedores

Un contenedor es un silo aislado y ligero para ejecutar una aplicación en el sistema operativo host.

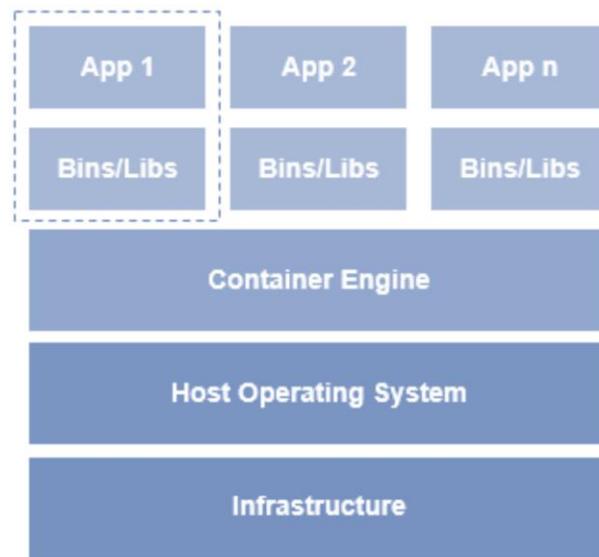
Los contenedores se basan en el kernel del sistema operativo host (que puede considerarse como la fontanería del sistema operativo) y solo contienen aplicaciones y algunas API y servicios de sistema operativo ligeros que se ejecutan en modo de usuario.

### Arquitectura de máquinas virtuales

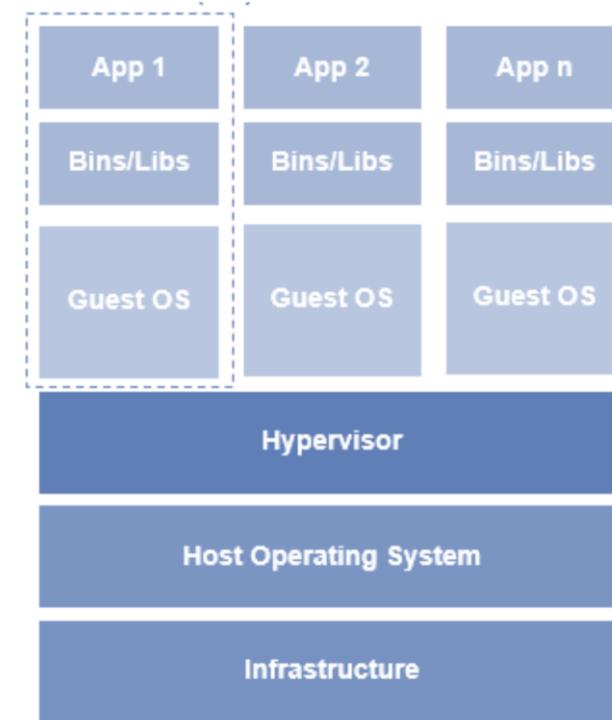
A diferencia de los contenedores, las máquinas virtuales ejecutan un sistema operativo completo, incluido su propio kernel.

## Virtualizaciones

### Arquitectura de contenedores

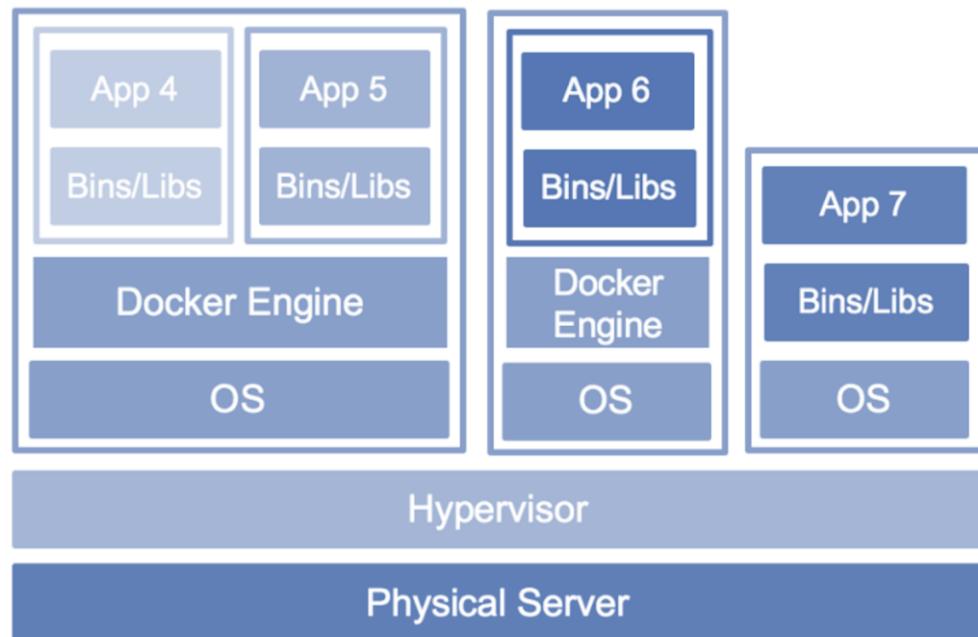


### Arquitectura de máquinas virtuales

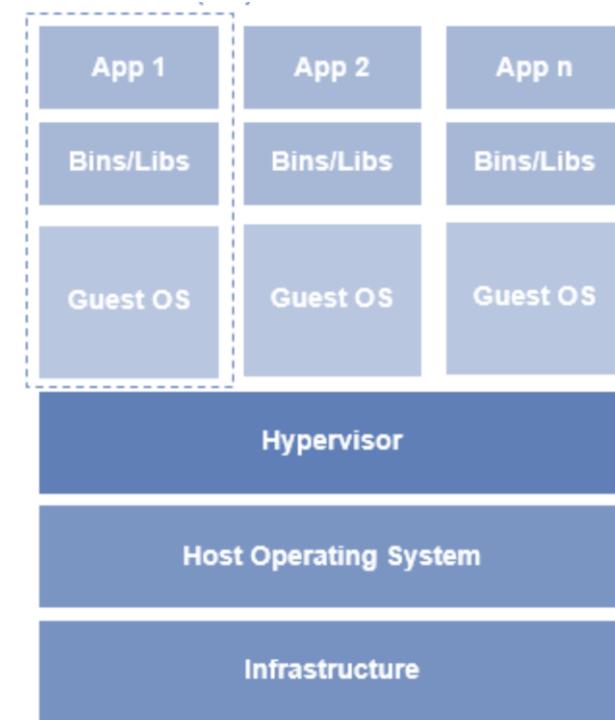


## Virtualizaciones

### Arquitectura de contenedores



### Arquitectura de máquinas virtuales



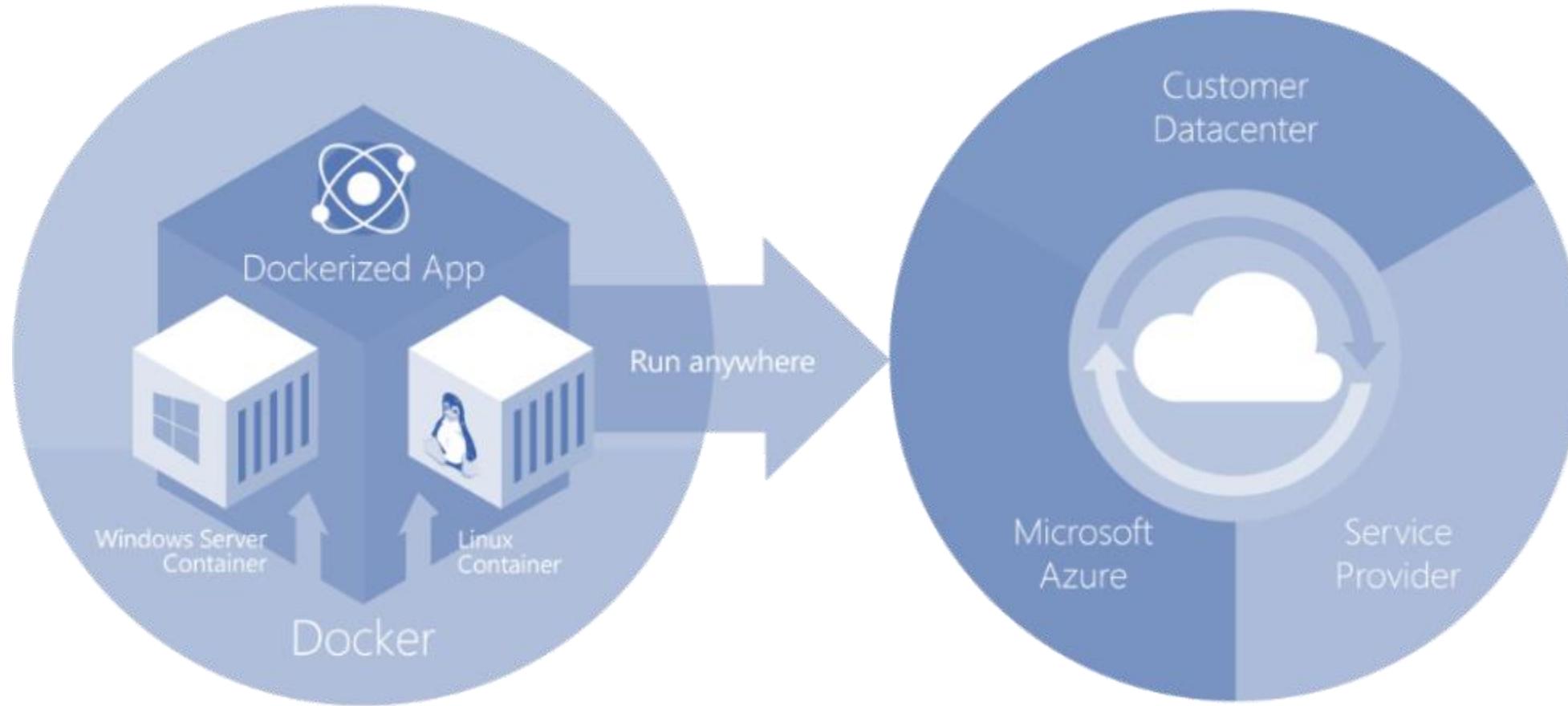
## Docker



Docker es un proyecto de código abierto para automatizar la implementación de aplicaciones como contenedores portátiles y autosuficientes que se pueden ejecutar en la nube o localmente. Docker es también una empresa que promueve e impulsa esta tecnología, en colaboración con proveedores de la nube, Linux y Windows, incluido Microsoft.

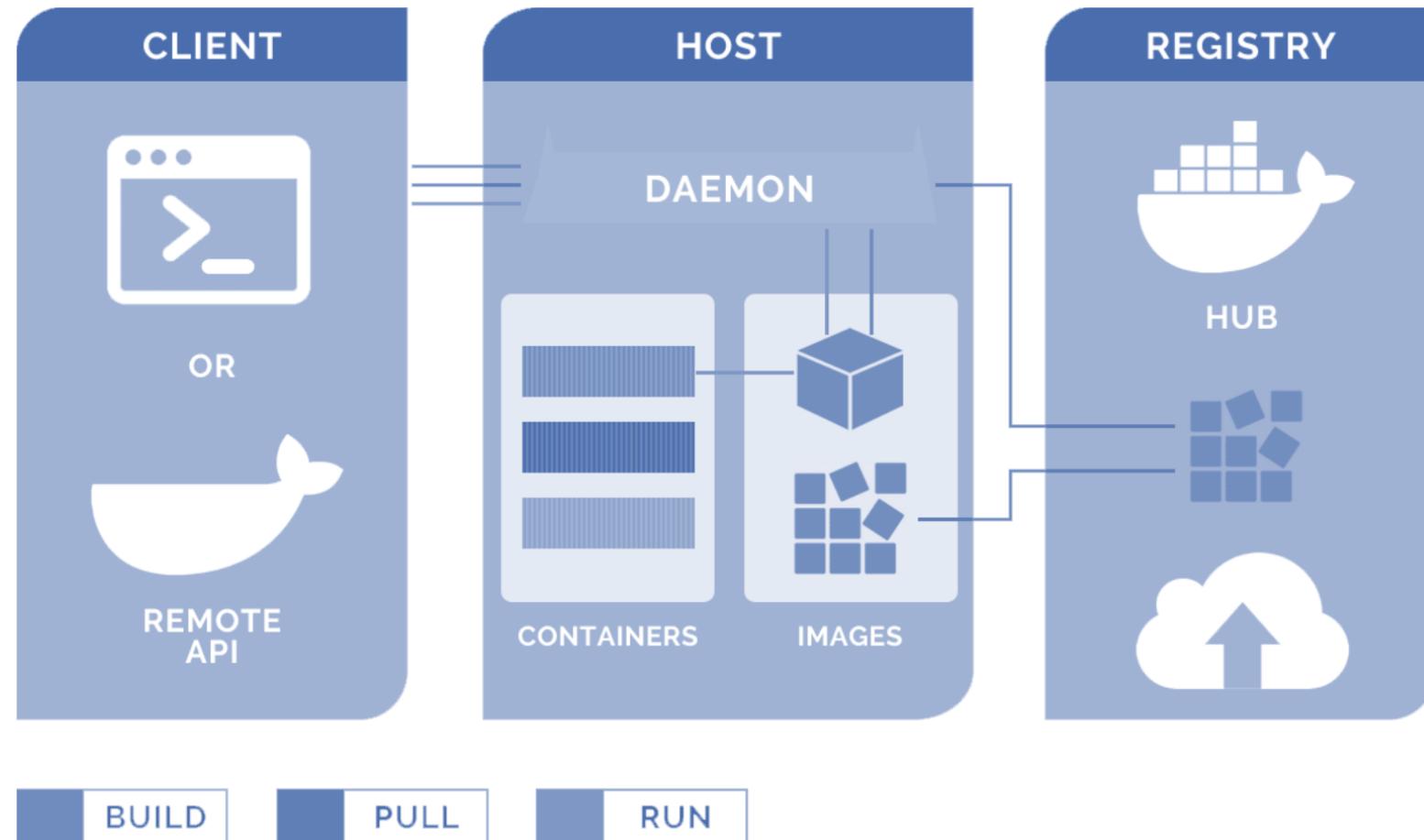
# → Contenerización

## Docker



# → Contenerización

## Arquitectura Docker

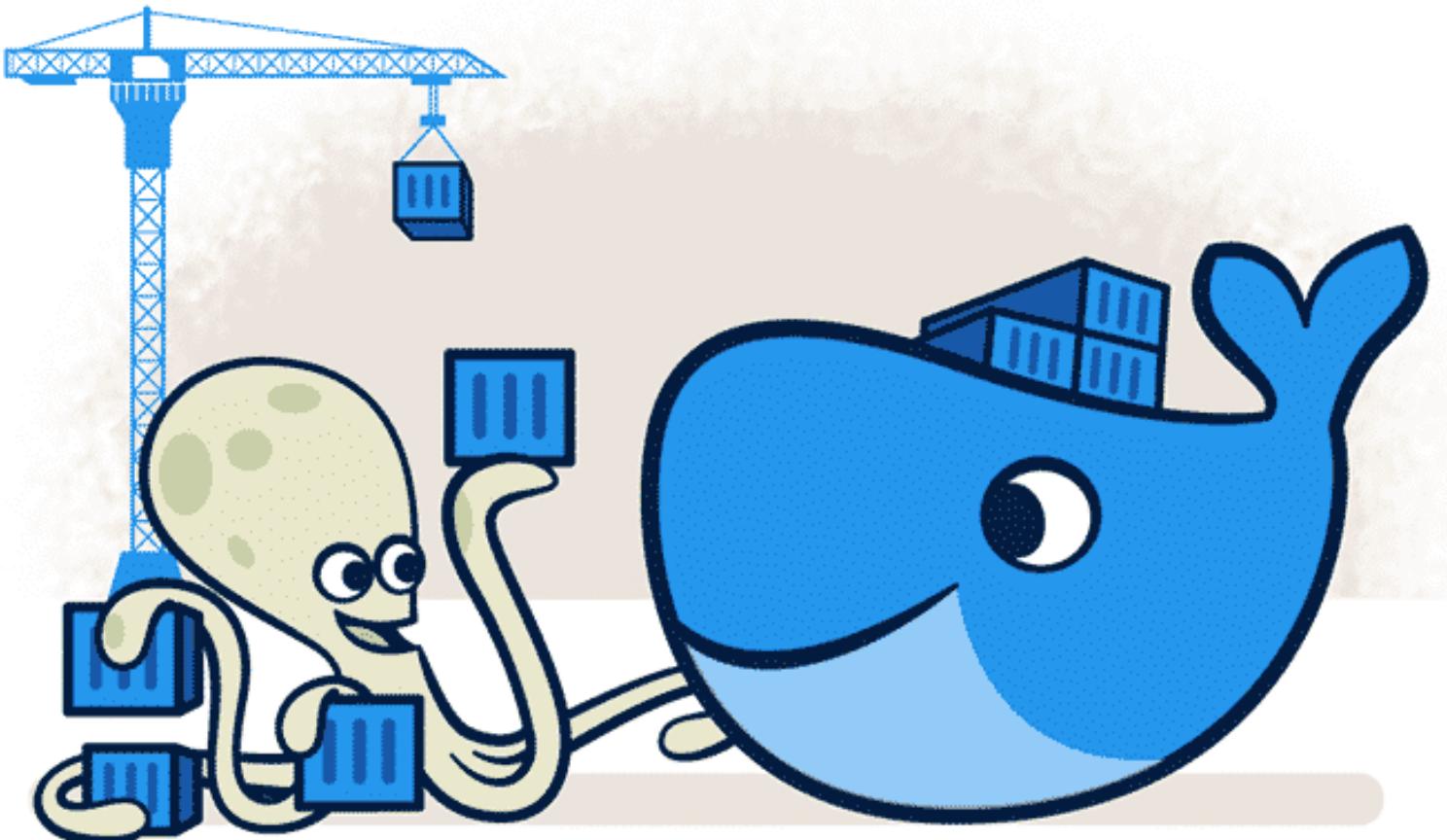


# 04



## Mi primer microservicio en .NET 6.

# Mi primer microservicio en .NET 6.





**GRACIAS**  
**POR SU PREFERENCIA**

